Relational Data Across Mathematical Libraries

Andrea Condoluci[1], Michael Kohlhase[2], Dennis Müller[2], Florian Rabe[2,3], Claudio Sacerdoti Coen[1], and Makarius Wenzel[4⋆]

[1] Università di Bologna
[2] Computer Science, FAU Erlangen-Nürnberg
[3] LRI, Université Paris Sud
[4] https://sketis.net

Abstract. Formal libraries are treasure troves of detailed mathematical knowledge, but this treasure is usually locked into system- and logic-specific representations that can only be understood by the respective theorem prover system. In this paper we present an ontology for using relational information on mathematical knowledge and a corresponding data set generated from the Isabelle and Coq libraries. We show the utility of the generated data by setting a relational query engine that provides easy access to certain library information that was previously hard or impossible to determine.

1 Introduction and Related Work

Overview and Contribution 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. However, integrating these libraries, or even reusing or searching in a single library can be very difficult because it currently requires understanding both the formal logic underlying it and the proof assistant used to maintain it.

We support these goals by extracting from the libraries semantic web-style relational representations, for which simple and standardized formalisms such as OWL2 [MPPS09], RDF [RDF04], and SPARQL [W3c] as well as highly scalable tools are readily available. Now it is well-known that relational formalisms are inappropriate for symbolic data like formulas, algorithms, and proofs. But our key observation is that if we systematically abstract all symbolic data away and only retain what can be easily represented relationally, we can already realize many benefits of library integration, search, or reuse.

Concretely, our contribution is threefold. Firstly, in §2, we design ULO, an upper library ontology for mathematical knowledge. We make ULO available as OWL2 XML file and propose it as a standard ontology for exchanging high-level information about mathematical libraries.

⋆ The authors were supported by DFG grants RA-18723-1 and KO-2428/13-1 OAF and EU grant Horizon 2020 ERI 676541 OpenDreamKit.

Secondly, in §3, we generate ULO data from concrete libraries in RDF format. For this paper we restrict ourselves to Coq and Isabelle as representative example libraries. Both datasets are massive, resulting in $\approx 10^7$ RDF triples each, requiring multiple CPU-hours to generate. We have OMDoc/MMT exports for about a dozen other libraries, including Mizar, HOLLight, TPS, PVS, from which we can generate ULO exports as well, but leave that to future work.

Thirdly, we demonstrate how to leverage these lightweight, high-level representations in practice. As an example application, in §4, 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).

Other applications enabled by our work include, e.g., graph-based visualization, cross-referencing between libraries, or integrating our formal library metadata with other datasets such as publication metadata or Wikidata.

Related Work The problem of retrieving mathematical documents that contain an instance or a generalization of a given formula has been frequently addressed in the literature [GC16]. The main difficulty is the fact that the formula structure is fundamental, but at the same time the matching must be up to changes to this structure (e.g. permutation of hypothesis, re-arrangement of expressions up to commutativity and associativity).

One solution is the technique presented in [Asp+06; AS04] that was applied to the Coq library. It consists in computing RDF-style triples that described the formula structure approximately, so that instantiation is captured by the subset relation of set of triples and matching up-to structural changes comes for free because the triples only record approximate shapes. Such a description is completely logic-independent, can be applied as well to other systems, and can be integrated with constraints over additional triples (e.g. over keywords, author, dependencies, etc). The Whelp search engine implemented the technique but is no longer maintained.

[Lan11] explores using the linked open data language and tool stack for representing mathematical knowledge. In particular, it presents various OWL ontologies, which are subsumed by the ULO, but does not have the data exports for the theorem prover libraries, which severely limited the reach of applications.

General purpose query languages for mathematical libraries data were previously introduced in [ADL12; Rab12]. Our experience is that the key practicality bottleneck for such languages is not so much the detailed definition of the language but the availability of large datasets for which querying is implemented scalably. This is the idea behind the approach we take here.

2 ULO: 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 §3), 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 [ULO] for the full documentation. For each concept, little icons indicate whether our extractors for Isabelle "" and Coq "" provide at least partial support (see also §3).

2.1 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[5], or MSC[6] and ACM[7] 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.

2.2 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 1 gives an overview of the most important classes in the ULO.

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.

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

`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 nonEmpty; `ulo:statement`[8] 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 §2.3.

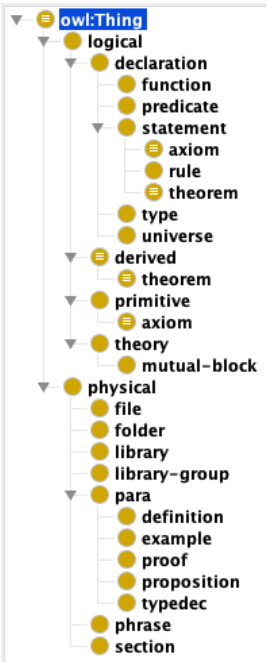


Fig. 1. ULO Classes

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

[8] 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.

- `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` 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).

2.3 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 [Asp+06] 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 occurrenes 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).

3 Exporting ULO Data from Prover Libraries

3.1 Exporting from Isabelle

Overview Isabelle 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. User contributions are centrally maintained in AFP, the *Archive of Formal Proofs* (`https://www.isa-afp.org`): this will provide substantial example material for the present paper (see §3.3).

The foundations of Isabelle due to Paulson [Pau90] are historically connected to *logical frameworks* like Edinburgh LF: this fits nicely to the LF theory of

MMT [RK13]. The Isabelle/MMT command-line tool [Wen18b] exports the λ-calculus of Isabelle/Pure into MMT as LF terms, with some add-on structures.

From a high-level perspective, Isabelle is better understood as *document-oriented proof assistant* or *document preparation system* for domain-specific formal languages [Wen18a]. 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.

PIDE is an approach by Wenzel to expose *aspects* of the ML language environment to the outside world, with the help of the Isabelle/Scala library for "Isabelle system programming". A major application of Isabelle/Scala/PIDE is Isabelle/jEdit, which is a Java-based text editor that has been turned into a feature-rich Prover IDE over 10 years of development [Wen18c]. To implement Isabelle/MMT [Wen18b], Wenzel has upgraded the Headless PIDE server of Isabelle2018 to support theory exports systematically. The Isabelle/MMT 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.

Isabelle2019 exports *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. Isabelle/MMT (repository version e6fa4b852bf9) turns this content into OMDoc and RDF/XML. This RDF/XML extractor supports both DC (Dublin Core Meta Data) and our ULO ontology (§2).

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 subsequent URI format that essentially consists of a triple (*long-theory-name*, *entity-name*, *entity-kind*):

`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.

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 [Wen18a],

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*. By writing "✎⟨*marker*⟩", the presentation context of a command may be augmented by arbitrary user-defined marker expressions. Isabelle/Pure already provides `title`, `creator`, `contributor` etc. from §2.3: 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 ✎⟨`contributor` ...⟩ sporadically. For automatic marking, metadata of AFP entries is re-used for their theories. One could also digest comments in theory files about authors, but this is presently unimplemented.

3.2 Exporting from Coq

Coq is one of the major interactive theorem provers in use. Many large libraries have been developed for Coq, covering both mathematics (e.g. the MathComp library that includes the proof of Feit-Thompson theorem; the CoRN library that covers many results in constructive analysis) and computer science (e.g. the proof of soundness of the CompCert compiler; the Color library about rewriting theory). We discuss some architectural choices for the extraction of RDF triples.

Libraries and URIs In contrast to Isabelle/AFP, there is no centralized maintenance of Coq libraries. There is even no index of publicly accessible libraries, even if many are nowadays hosted on GitHub or at least have a downloadable tarball. Moreover, Coq does not even has a proper notion of library: the Coq compiler processes individual `.v` files and a library is usually a bunch of Coq files together with a `Makefile` to compile them in the right order. However, Coq has a notion of *logical* names: when a file is compiled, the compiler is invoked passing a logical name like `mathcomp.field` and every object declared in the file will be given a logical name whose prefix is `mathcomp.field`. For technical reasons (e.g. to address sub-objects or things that are not Coq objects) the URIs we use are not logical names, but we try to keep a correspondence where possible. For example `cic:/mathcomp/field/falgebra/SubFalgType/A.var` is the URI of a variable declared into the `SubFalgType` section of the file `falgebra.v` compiled with logical name prefix `mathcomp.field`.

Opam packages There is a recent effort by the Coq team to push developers of libraries to release *opam* packages for them. Opam is a package manager for ocaml libraries that can be (ab)used to automatically download, compile and install Coq libraries as well. Moreover, to release an opam package some Dublincore like metadata like author and synopsis must be provided. Other interesting mandatory metadata are license and version. Finally, opam packages specify the exact Coq version they depend on, granting that compilation will succeed.

To make Coq libraries accessible to other tools, Sacerdoti Coen wrote a fork of Coq 8.9.0[9] (the current stable release) that can be automatically invoked by opam and that behaves exactly as the standard Coq, but for the fact that it produces multiple XML files that describe the content of the Coq library. The XML files encode the information present in Coq kernel augmented with additional data coming from the sources or computed when the Coq sources are elaborated. In the remainder of the paper we identify the notion of library (that Coq lacks) with that of an opam package: all libraries without a package will be ignored. The exported files are collected in Git repositories in one-to-one correspondence with opam packages[10].

Coqdoc output Coq comes with a standard tool, named `coqdoc`, to automatically generate a Web site that documents a library. The Web pages contain pretty-printed and syntax highlighted copies of the sources where additionally hyperlinks are introduced for every identifier defined in the library. In particular, each object in the library is given an HTML anchor in some HTML page. Finally, the pages also include markup automatically generated from special comments that the user adds to the source files.

After extracting opam packages to XML we run coqdoc over the union of all the extracted libraries, obtaining the Web site that documents all the exported libraries (available at `https://coq.kwarc.info/`).

RDF triples We generate RDF triples from three different sources. The first source is the description of the opam packages. Each package is given a URI that mangles its name and version. Triples map this URI to the available opam metadata.

The second source are the (compressed) XML files exported by Sacerdoti Coen's fork [Sac]. In particular we run Python scripts over the XML files to collect all the ULO triples related to Coq objects (definitions, theorems, modules, sections, etc.). Each object is represented on disk either as a directory (if it contains other objects) plus additional XML files (to attach additional data) or to an XML file on disk (if it is atomic). The physical structure on the filesystem is exactly the URI structure: the file `A.var.xml.gz` whose URI is `cic:/mathcomp/field/falgebra/SubFalgType/A.var` is stored in the `mathcomp/field/falgebra/SubFalgType` directory of the `coq-mathcomp-field-1.7.0` Git repository. The repository was generated exporting from the opam package `coq-mathcomp-field`, version 1.7.0. The scripts themselves are therefore quite straightforward: for each Git package, they just recursively traverse the filesystem and the XML trees collecting the triples and adding them to the repository.

The third source is the coqdoc generated website: `ulo:docref` maps URIs to relative URLs pointing to website. E.g. `cic:/mathcomp/field/falgebra/SubFalgType/A.var` is mapped to `mathcomp.field.falgebra.html#SubFalgType.A`.

[9] `https://github.com/sacerdot/coq`
[10] `https://gl.mathhub.info/Coqxml`

Precision The ULO ontology is useful as long as it is reused for different systems and it is the result of a compromise. For instance, multiple structuring notions like Coq modules, functors, module types and sections are all mapped to `ulo:theory`. It is in principle possible to also export Coq-specific triples to run Coq-specific queries, but we have not followed this direction.

Coverage There is a certain number of ULO relations that are currently not generated for Coq. We classify them into three classes. The first one is information that is inferrable from the XML sources, but requires non-trivial computations (e.g. computing the type of some lambda-term to decide if it encodes a proof via Curry-Howard or otherwise is a proper term). The second class is information that is not recorded in the XML files but that could be recorded modifying the XML exporter (e.g. `ulo:external-size`, `ulo:check-time` or `ulo:simplification-rule`). The third class is information that must be user provided (e.g. `ulo:similar-to`, `ulo:formalizes` or `ulo:aligned-with`) and that is completely absent from Coq sources.

Future work As future work we plan to improve the Coq XML exporter and the RDF scripts to achieve full coverage of the first two classes. To cover the third class, we would badly need an extension of the input language of Coq to let the user add machine-understandable metadata to the sources, like Isabelle does. The extension would need to be official accepted upstream and adopted by users before information belonging to the third class can be exported automatically.

3.3 Statistics

Here are some statistics for both Isabelle[11] and Coq, referring to various subsets of the available libraries. 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` and `https://gl.mathhub.info/Coqxml/coq.8.9.0`.

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
All 49 Libraries	383,527	11,516,180	1,979	-	6,061	-	161,736	452 MB	

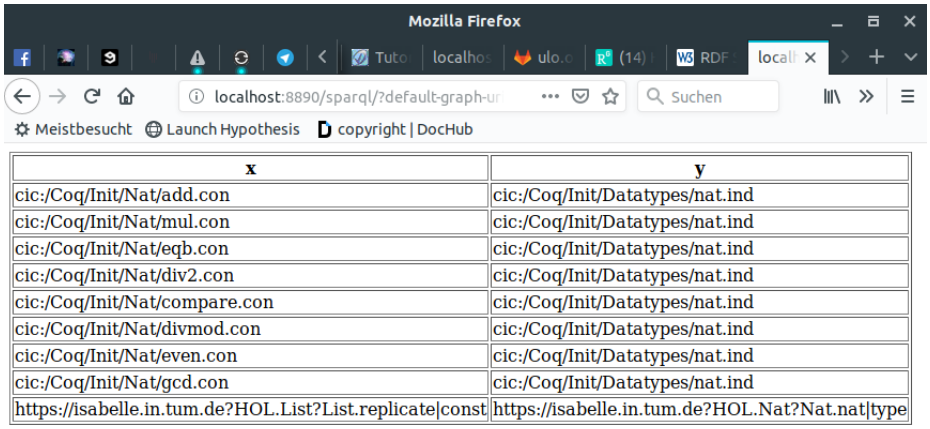
4 Applications

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[12], which reads the exports described in Sect. 3 and provides a web interface with a SPARQL

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

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



x	y
cic:/Coq/Init/Nat/add.con	cic:/Coq/Init/Datatypes/nat.ind
cic:/Coq/Init/Nat/mul.con	cic:/Coq/Init/Datatypes/nat.ind
cic:/Coq/Init/Nat/eqb.con	cic:/Coq/Init/Datatypes/nat.ind
cic:/Coq/Init/Nat/div2.con	cic:/Coq/Init/Datatypes/nat.ind
cic:/Coq/Init/Nat/compare.con	cic:/Coq/Init/Datatypes/nat.ind
cic:/Coq/Init/Nat/divmod.con	cic:/Coq/Init/Datatypes/nat.ind
cic:/Coq/Init/Nat/even.con	cic:/Coq/Init/Datatypes/nat.ind
cic:/Coq/Init/Nat/gcd.con	cic:/Coq/Init/Datatypes/nat.ind
https://isabelle.in.tum.de?HOL.List?List.replicate|const	https://isabelle.in.tum.de?HOL.Nat?Nat.nat|type

Fig. 2. Virtuoso Output for the Example Query using Alignments

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 [Mül+17] 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 2.

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, 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:uses ?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 = <cic:/Bignums/BigN/BigN/BigNring.con> )
  }
ORDER BY ?dist DESC 2
```

The above code queries for all symbols recursively used in the (effectively randomly chosen) Lemma `BigNring` stating that the ring of arbitrary large natural numbers in base 2^{31} is a semiring; the output for that query is shown in Figure 3.

o	dist
cic:/Coq/setoid_ring/Ring_theory/semi_ring_theory.ind	1
cic:/Bignums/BigN/BigN/BigN/mul_comm.con	1
cic:/Bignums/BigN/BigN/BigN/mul_assoc.con	1
cic:/Bignums/BigN/BigN/BigN/mul_add_distr_r.con	1
cic:/Bignums/BigN/BigN/BigN/mul_1_l.con	1
cic:/Bignums/BigN/BigN/BigN/mul_0_l.con	1
cic:/Bignums/BigN/BigN/BigN/add_comm.con	1
cic:/Bignums/BigN/BigN/BigN/add_assoc.con	1
cic:/Bignums/BigN/BigN/BigN/add_0_l.con	1
cic:/Bignums/BigN/BigN/BigN/one.con	1
cic:/Bignums/BigN/BigN/BigN/zero.con	1
cic:/Bignums/BigN/BigN/BigN/t.con	1
cic:/Bignums/BigN/BigN/BigN/mul.con	1
cic:/Bignums/BigN/BigN/BigN/eq.con	1
cic:/Bignums/BigN/BigN/BigN/add.con	1
cic:/Coq/setoid_ring/Ring_theory/DEFINITIONS/R.var	2
cic:/Coq/setoid_ring/Ring_theory/DEFINITIONS/rO.var	2
cic:/Coq/setoid_ring/Ring_theory/DEFINITIONS/rmul.var	2
cic:/Coq/setoid_ring/Ring_theory/DEFINITIONS/rI.var	2
cic:/Coq/setoid_ring/Ring_theory/DEFINITIONS/req.var	2
cic:/Coq/setoid_ring/Ring_theory/DEFINITIONS/radd.var	2
cic:/rO.var	2
cic:/rmul.var	2
cic:/rI.var	2
cic:/radd.var	2
cic:/req.var	2
cic:/R.var	2

Fig. 3. 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 [ADL12]. Instead [Asp+06; AS04] 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.

5 Conclusion and Future Work

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. Consequently, future work will be strongly community-based.

We envision 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 the libraries of Isabelle and Coq. 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, using our publicly available datasets for Isabelle and Coq, 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.

References

[ADL12] D. Aspinall, E. Denney, and C. Lüth. "A Semantic Basis for Proof Queries and Transformations". In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by N. Bjørner and A. Voronkov. Springer, 2012, pp. 92–106.

[AS04] Andrea Asperti and Matteo Selmi. "Efficient Retrieval of Mathematical Statements". In: *Mathematical Knowledge Management, MKM'04*. Ed. by Andrea Asperti, Grzegorz Bancerek, and Andrej Trybulec. LNAI 3119. Springer Verlag, 2004, pp. 1–4.

[Asp+06] Andrea Asperti et al. "A Content Based Mathematical Search Engine: Whelp". In: *Types for Proofs and Programs, International Workshop, TYPES 2004, revised selected papers*. Ed. by Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner. LNCS 3839. Springer Verlag, 2006, pp. 17–32.

[GC16] Ferruccio Guidi and Claudio Sacerdoti Coen. "A Survey on Retrieval of Mathematical Knowledge". In: *Mathematics in Computer Science* 10.4 (2016), pp. 409–427. DOI: 10.1007/s11786-016-0274-0.

[Lan11] Christoph Lange. *Enabling Collaboration on Semiformal Mathematical Knowledge by Semantic Web Integration*. Studies on the Semantic Web 11. Heidelberg and Amsterdam: AKA Verlag and IOS Press, 2011.

[MPPS09] Boris Motik, Bijan Parsia, and Peter F. Patel-Schneider. *OWL 2 Web Ontology Language: XML Serialization*. W3C Recommendation. World Wide Web Consortium (W3C), Oct. 27, 2009. URL: http://www.w3.org/TR/2009/REC-owl2-xml-serialization-20091027/.

[Mül+17] Dennis Müller et al. "Classification of Alignments between Concepts of Formal Mathematical Systems". In: *Intelligent Computer Mathematics (CICM) 2017*. Ed. by Herman Geuvers et al. LNAI 10383. Springer, 2017. DOI: 10.1007/978-3-319-62075-6.

[Pau90] Lawrence C. Paulson. "Isabelle: The Next 700 Theorem Provers". In: *Logic and Computer Science*. Ed. by P. Odifreddi. Academic Press, 1990, pp. 361–386.

[Qed] *The QED Project*. http://www-unix.mcs.anl.gov/qed/. 1996. URL: http://www-unix.mcs.anl.gov/qed/.

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

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

[ULO] *ULO Documentation*. URL: https://ulo.mathhub.info/ (visited on 03/11/2019).

[W3c] *SPARQL 1.1 Overview*. W3C Recommendation. World Wide Web Consortium (W3C), Mar. 23, 2013. URL: https://www.w3.org/TR/sparql11-overview/.

[Wen18a] Makarius Wenzel. "Isabelle/jEdit as IDE for domain-specific formal languages and informal text documents". In: *F-IDE 2018 – 4th Workshop on Formal Integrated Development Environment*. Ed. by Paolo Masci, Rosemary Monahan, and Virgile Prevosto. 2018. URL: https://sketis.net/wp-content/uploads/2018/05/isabelle-jedit-fide2018.pdf.

[Wen18b] Makarius Wenzel. *Isabelle/MMT: export of Isabelle theories and import as OMDoc content*. Blog entry. https://sketis.net/2018/isabelle-mmt-export-of-isabelle-theories-and-import-as-omdoc-content. 2018.

[Wen18c] Makarius Wenzel. *Isabelle/PIDE after 10 years of development*. Presented at the UITP workshop: User Interfaces for Theorem Provers. https://sketis.net/wp-content/uploads/2018/08/isabelle-pide-uitp2018.pdf. 2018.

[RDF04] RDF Core Working Group of the W3C. *Resource Description Framework Specification*. http://www.w3.org/RDF/. 2004.

[Sac] Claudio Sacerdoti Coen. "A Coq plugin to export its libraries to XML". Submitted to CICM 2019. URL: http://www.cs.unibo.it/~sacerdot/cicm19/xmlexport.pdf.