Making Isabelle Content Accessible in Knowledge Representation Formats

Michael Kohlhase
University Erlangen-Nürnberg

Florian Rabe
University Erlangen-Nürnberg

Makarius Wenzel
https://sketis.net, Augsburg, Germany

Abstract

The libraries of proof assistants like Isabelle, Coq, HOL are notoriously difficult to interpret by external tools: de facto, only the prover itself can parse and process them adequately. In the case of Isabelle, an export of the library into a FAIR (Findable, Accessible, Interoperable, and Reusable) knowledge exchange format was already envisioned by the authors in 1999 but had previously proved too difficult.

After substantial improvements of the Isabelle Prover IDE (PIDE) and the OMDoc/Mmt format since then, we are now able to deliver such an export. Concretely we present an integration of PIDE and Mmt that allows exporting all Isabelle libraries in OMDoc format. Our export covers the full Isabelle distribution and the Archive of Formal Proofs (AFP) — more than 12 thousand theories and locales resulting in over 65 GB of OMDoc/XML.

Such a systematic export of Isabelle content to a well-defined interchange format like OMDoc enables many applications such as dependency management, independent proof checking, or library search.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Isabelle, PIDE, OMDoc, MMT, library, export

Digital Object Identifier 10.4230/LIPIcs...I

1 Introduction and Related Work

Motivation A critical bottleneck in the field of interactive theorem proving is the lack of interoperability between proof assistants and related tools. This leads to a duplication of efforts: both formalizations and auxiliary tool support (e.g., for automated proving, library management, user interfaces) cannot be easily shared between systems. This situation is well-understood by the community and has persisted for decades despite occasional attempts to achieve interoperability by standardization or library translations.

The story of this article started in 1999, when one author (Kohlhase, who worked on the OMDoc interchange format [27] for formal libraries) wrote an email to another one (Wenzel, who worked on the Isabelle proof assistant [43, 44]) asking about the status of ongoing efforts to export Isabelle theories in some format that could be further transformed into OMDoc. Just 19 years later, Wenzel replied to the same email announcing that an Isabelle→OMDoc export now works routinely. Critically, this export was enabled by the PIDE and Mmt infrastructures developed for Isabelle by Wenzel resp. for OMDoc by Rabe in the interim. Despite this massive groundwork laid in the last two decades, the export itself still required about 9 person-months to implement. This paper tells the story of how we achieved this export after such a long time.

Isabelle99 (October 1999) was a rather small experimental proof assistant for multiple object logics, with ≈ 1 MB source text the for Isabelle/ZF library and ≈ 3 MB for Isabelle/HOL.

The ZF library was particularly interesting for Kohlhase at that time and considered large. In contrast, Isabelle2020 (April 2020) includes ≈ 2 MB material for ZF and ≈ 30 MB for HOL, or rather ≈ 160 MB if the Archive of Formal Proofs (AFP) is included. The PIDE/MMT work flow described in this paper requires a server-class machine to handle all this material: 80 GB RAM, 8 CPU cores, and 22 h elapsed time (this includes theory and proof processing by Isabelle). Thus, a major portion of publicly known Isabelle content[1] becomes accessible as XML in the OMDOC format: 65 GB uncompressed or 300 MB with XZ compression.

Related Work In both formalizations and auxiliary tool support, previous work has shown significant potential for knowledge sharing. Regarding sharing among proof assistants, library translations such as [41, 23, 26, 34] have been used to transport theorems across systems. An unusual approach is virtualization of HOL4 in Isabelle [18], where the ML environment of Isabelle is carefully instrumented to load the HOL4 library sources (also in ML) and reconstruct theories and proofs within the Isabelle/Pure inference kernel.

Most of these approaches produce an isolated image of the source library within the target library. Alignments [21] have been used to match pragmatically corresponding concepts defined in different libraries [10]. In contrast, [18] connects interesting results via *lifting and transfer*, where only the signatures of the main conclusions need to be taken into account.

Regarding sharing among proof assistants and auxiliary tools, Isabelle/Sledgehammer [35, 44] integrates different automation tools generically, and Dedukti [7] has been used as independent checker for various proof assistant libraries. Premise selection tools use, e.g., machine-learning [22], to reduce the search space when running automated provers on subgoals. In all cases, a single tool could be used for every proof assistant — provided the language and library are available in a universal format that can be plugged into it.

Unfortunately, the latter point — the universal format — is often prohibitively expensive for many interesting applications. Firstly, it is extremely difficult to design a format that strikes a good trade-off between simplicity and universality. And secondly, even in the presence of such a format, it is difficult to implement the export of a library into it. Here it is important to realize that any export attempt is doomed that uses a custom parser or type checker for the library — only the internal data structures maintained by the proof assistant are informative enough for most use cases. Consequently, only expert developers can perform this step, and of these, each proof assistant community only has very few.

In previous work, the authors have developed such a universal format [27, 48, 29] for formal knowledge: OMDOC is an XML language geared towards making formula structure and context dependencies explicit while remaining independent of the underlying logical formalism. We also built a strong implementation — the MMT system — and a number of generic services, e.g., [46, 30]. In the DFG-funded OAF Project (Open Archive of Formalization), we have developed export for Mizar [17], HOL Light [24], IMPS [6], PVS [28], and Coq in [38]. In what we now call the *OAF approach*, we systematically

(i) defined the logic of the proof assistant in a logical framework by hand,
(ii) instrumented the proof assistant to export its libraries, and
(iii) use the instrumented prover to export the libraries

for all these exports. MMT provides the semantics that ties together the three involved levels (logical framework, logic, and library) and provides a uniform high-level API for further

[1] In the Isabelle community, contributions are usually submitted to AFP for long-term maintenance, and thus become centrally accessible. Only a few exceptional projects are maintained independently (e.g. seL4 `https://sel4.systems` or IsaFoR `http://cl-informatik.uibk.ac.at/isafor`).

processing. [32] gives on overview over the theoretical, technical, and social challenges of the OAF exports.

In the work reported here, we follow this basic recipe with a few modifications. Firstly, because Isabelle already includes a logical framework, we do not encode Isabelle in yet another one. Instead, we extend the existing LF formalization in Mmt to obtain one for the Pure framework of Isabelle. There are two reasons for this choice: it is conceptually appropriate as it puts the logics defined in Isabelle on the same levels as those defined in other logical frameworks (e.g., Mmt/LF/HOL Light and Mmt/Isabelle/HOL); it also improves scalability by avoiding another layer of logical framework-encoding. Secondly, Isabelle is extremely complex, and a large portion of our work went to streamlining Isabelle components to enable step (ii) above, notably the Isabelle PIDE infrastructure for incremental processing of proof documents. Thirdly, the resulting exports of the Isabelle libraries were significantly larger than any exports we had handled previously. Therefore, we had to develop new optimizations both on the Isabelle and on the Mmt side to be able to carry out step (iii) above.

Repeating such an advanced Mmt integration for other proof assistants must revisit the particular technology found there. In particular, proof assistants can vary widely in how the building of large projects and of dependencies between projects are handled. For example, Coq uses a decentralized library with hundreds of repositories and consequently uses sophisticated tools for repository management and continuous integration, e.g., the piCoq tool [42] to manage build processes in a fine-grained manner. Thus, the corresponding problem is more complex for Coq as it is for Isabelle, where the library is more centralized and the build management is tightly integrated with the kernel. piCoq already involves some Java-based components, which might help integrate it with the Mmt Scala API.

Contribution and Overview We apply our approach to Isabelle [44]: we present a definition of the Isabelle logical framework in Mmt and an export feature for Isabelle logics and libraries. We exemplify the latter by exporting the standard Isabelle distribution [19] and the *Archive of Formal Proofs* [1]. The translated libraries are available at `https://gl.mathhub.info/Isabelle` as compressed OMDoc files.

We present preliminaries about Isabelle and PIDE as well as OMDoc and Mmt in Sections 2 and 3. Then we describe the logical and the technical aspects of the export in Sections 4 and 5. We sketch some applications enabled by the export in Section 6.

It is difficult to estimate the total workload covered by this paper because it builds on decades of implementation work in both Isabelle and Mmt, much of which was never published in itself. But concretely for this particular export, we spent about 1 person-month on the overall design of the translation and the implementation, 6 person-months on the implementation on the Isabelle side, 1 on the Mmt side, and 1 on administrative parts and dissemination of the results.

Acknowledgments The authors were supported by DFG grant RA-18723-1 OAF and EU grant Horizon 2020 ERI 676541 OpenDreamKit.

2 Isabelle and PIDE

The Isabelle Platform Isabelle [43, 44] is a generic platform for formal logic tools. Its foundation is the *Pure logical framework* by Paulson [43] based on a minimal intuitionistic higher-order logic with declarative natural deduction proofs. Isabelle/Pure is used to represent

object-logics like Isabelle/FOL, Isabelle/ZF, and the most widely used Isabelle/HOL based on Church's simple type theory and Gordon's HOL [11].

Extra-logical tools are implemented in the *Meta Language (ML)* in LCF style [12]. Isabelle/ML has full access to the symbolic representation of the logic and provides many add-ons such as concrete syntax and context management for proof tools. The ML compiler and toplevel environment are managed within the same formal context as the logic, so ML declarations follow the structure of theory specifications and proofs.

ML is mainly used for pure mathematical programming with limited access to the physical world. Additionally, Scala (running on the Java platform) is used for external tooling: it manages ML processes, formal sources, and the resulting content, and provides an outer shell for Isabelle systems programming with access to GUI frameworks, TCP servers, database engines, etc. The programming style of Isabelle/Scala resembles Isabelle/ML, and some important modules are available on both sides (e.g. formatting of pretty-printed text).

Isabelle's Prover IDE framework PIDE [49] integrates all development into the semantic text editor Isabelle/jEdit [52]. While the user is composing text, PIDE provides real-time markup about its meaning — rendered as, e.g., text color, squiggly underline, tooltips, hyperlinks, icons in the border. The Prover IDE supports ML development as well: users can edit theory sources with embedded ML modules directly, while the ML compiler does static checking and dynamic evaluation on the spot. Thus Isabelle has no need for externally compiled modules, in contrast to, e.g., Coq plugins.

More recently, Isabelle/PIDE has been refined to support *headless mode*, which lets a function in Isabelle/Scala observe this markup while a formal library is processed in Isabelle/ML. Compared to traditional batch-builds, headless PIDE provides more detailed feedback from the prover and more flexibility in dynamic loading and unloading of theories. In particular, it allows the processing of Isabelle content for other purposes than editing it in a GUI. This is the central interface that we use in the work reported in this article.

Isabelle Libraries The standard distribution of Isabelle includes the Isabelle/HOL library with many examples, but the bulk of applications is in the *Archive of Formal Proofs* (AFP), which is organized like a scientific online journal. In April 2020, AFP had 528 articles by 347 authors, comprising a total of 130 MB of source text in 5343 theory files.

Formal processing of the Isabelle distribution plus AFP requires ≈ 46h CPU time or 13h elapsed time, using standard hardware with 8 CPU cores and 16 GB RAM. Such `isabelle build` jobs [53] produce heap images for the internal state of Isabelle/ML and optional HTML/PDF documents that resemble conventional mathematical texts.

Library Structure Isabelle libraries consist of formal documents [50] structured according to session definitions, theory imports, and commands within theories:

- A *session* is a collection of theories with optional LaTeX document preparation. It may refer to a single *parent session* and multiple *import sessions* (to reuse some of their theories by reloading their sources within the original session name space). For example, the session `HOL` is the basis for most applications, and the session `HOL-Analysis` is a substantial library of standard mathematics. In the AFP, each entry (or "article") usually corresponds to a single session with its own setup for the published PDF document.
- A *theory* is a linear arrangement of commands corresponding to definition–statement–proof in conventional mathematical texts. The theory header imports multiple parent theories, taking a strictly monotonic *merge* of existing theories as basis for the new one.

For example, theories like `HOL.Nat`, `HOL.List` are stepping stones towards `Main` and `Complex_Main`, which have global names and are the key entry-points for applications.
- A *command* is a functional update on the theory context (or proof state) using concrete syntax within the source file. Command syntax may embed embed user-defined sublanguages delimited as so-called "cartouches", e.g. **ML** ‹*val a = 1*›. Theories may define new commands at any time — even Isabelle/Pure itself is defined in user-space relying only on the **ML** command for bootstrapping. For example, the commands **definition**, **inductive**, **fun** define constants and automatically prove characteristic theorems over them, while **lemma**, **proof**, **qed**, **by** are for proofs written in the Isar proof language.

The overall graph of sessions and theories is managed by Isabelle to exploit parallel processing within multithreaded ML (and Scala). For example, a theory could already be finished on the surface but some of its proofs still pending in parallel forks. Isabelle/Scala provides operations to explore sources down to command spans (keyword with argument tokens), without requiring a prover process to interpret them in the formal context.

Library Processing The library sources are processed by feeding them to the Isabelle/ML session managed by Isabelle/Scala. This constructs formal meaning that is a-priori opaque, i.e., a matter of the private context of the logic or user-defined sublanguage. In order to expose some aspects of the meaning, Isabelle/ML supports several formal message channels:

- *Output* of regular messages, warnings, errors, etc. with text that typically refers to logical types and terms. Pretty-printing with blocks and breaks is supported by default: the front-end usually does the formatting based on precise window and font sizes. For example, the command **term** turns its source argument into an internal term and pretty-prints the result with markup to link constants to their definitions.
- *Reports* to assign markup to existing input sources (with precise positions). For example, after reading a term from the source text its precise positions of free and bound variables are reported as XML markup elements `<free/>` and `<bound/>`. The editor turns this into the usual Isabelle color scheme of blue vs. green variables.
- *Exports* to attach arbitrary blobs to a theory (with hierarchic names separated by slash). For example, the command **export_code** turns Isabelle/HOL specifications into program source (for SML, OCaml, Scala), and the result becomes an export artifact of the enclosing theory. Thus the current version of input sources (e.g., an open buffer in Isabelle/jEdit) is augmented by the result of **export_code** seen as a mathematical function; the editor shows the result via the *virtual file-system* URL `isabelle-export:` within its File Browser, independently of the accidental state of the physical file-system.

The exposed aspects of document meaning are stored within the *session database*. For conventional batch-builds, that is an SQLite database file used like an archive with XZ-compressed entries, and the command-line tool `isabelle export` lists and extracts its content. For PIDE processing, the database consists of Scala values within the document snapshot and may be explored via user-provided Scala functions, e.g., for GUI painting of annotated document source. It is also possible to write out the data to another database (e.g., PostgreSQL is supported routinely), or in a completely different application, which is what we do in the OAF-style export reported on in this article.

To support the latter, Wenzel has modified the processing to allow for application-specific ML functions for presentation. Whenever a theory node with all its imports is fully consolidated (parallel proofs finished), user-defined ML functions can access its list of commands paired with the internal theory context at each step.

Isabelle/Pure and Isabelle/HOL provide standard presentation functions to expose core material from the logical context, guarded by option `export_theory`. Results are exported to the session database, using a private XML representation, Isabelle YXML transfer syntax, and XZ compression of the resulting blob. This works both for batch sessions (`isabelle build`) and for headless PIDE sessions (`isabelle dump`). Thus, with the current infrastructure, the request by Kohlhase from 1999 could be fulfilled on the spot via `isabelle dump -B ZF`, but instead of digesting raw XML/YXML data it is better to use typed APIs in Isabelle/Scala (by using module `Export_Theory` as we do in Section 5.1).

3 OMDoc and MMT

Language OMDoc [27] (short for **O**pen **M**athematical **Doc**uments) is a semantics-oriented XML-based markup format for STEM-related documents. It conceptualizes mathematical objects in three levels as seen in Figure 1: the *object* level for mathematical formulas and their presentations, the *statement* level for definitions, theorems, proofs, etc, and the *theory* level for collections of statements. Each level comes in two dimensions for the formal representations of the content addressed to mathematical software systems and the narrative structure addressed to humans. Higher levels may contain expressions of lower ones, and mixtures of dimensions are allowed, leading to a overall format that can handle flexible levels of formality (see [31] for a discussion).

Even at the early state in 1999, OMDoc already had this general architecture and was therefore well-suited in principle for representing Isabelle content, in particular the Isar proof language [54] that was new at the time. But the formal part of OMDoc was purely descriptive and lacked a rigorous semantics. In particular, the role of the logical systems needed for formally stating mathematical properties was almost fully unspecified beyond the idea — inherited from OpenMath — that logics are theories as well.

level	formal	narrative
object	OpenMath	presentation MathML
statement	sequents	paragraphs + cues
theory	theories/views	sections, etc.

Figure 1 Three level & two dimensions in OMDoc

Later Mmt (Meta Meta Theories) [48] re-conceptualized and refined the formal fragment of OMDoc, greatly enhancing both rigor and expressivity. It models formal objects and statements using logical frameworks, in particular the judgments-as-types paradigm, and bases OMDoc's theory level on the category of theories and theory morphisms following the development graphs approach [2]. The former allows for fine-grained specifications of the semantics of individual objects, and the latter allows for inducing and translating knowledge across theories. A new *meta-theory* relation links a logical framework to the logics defined in it, thus formalizing the "logics-as-theories" approach.

The Mmt System The OMDoc/Mmt language is implemented in the Mmt system (Meta Meta Toolset; see [47]), which provides an API for the language constructs at all levels and provides both logical services such as type reconstruction and rewriting and knowledge management services such as IDE and HTML presentation and browsing of libraries.

Because it avoids committing to a specific semantics or logical foundation, foundation-dependent services and features (e.g., type reconstruction) are implemented by splitting the algorithms into a foundation-independent kernel that is user-extensible with foundation-specific rules. For example, the logical framework LF [15] is implemented using about 10 rules taking only a few lines of code each.

Theory Graphs Theory graphs are diagrams in the categories of theories and morphisms. The possible morphisms in MMT are **inclusions**, which import all declarations from the domain to the co-domain, **structures**, which are like includes but copy and translate all declarations, **views**, which are semantics-preserving translations from domain to codomain, and the **meta-theory**-relation, which behaves like an include for most purposes.

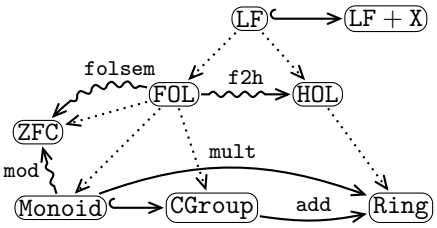

Figure 2 Meta-Levels in OMDoc/MMT

Figure 2 shows an example of a typical setup of formalizations in MMT: Dotted lines represent the meta-theory-relation, hooked arrows are includes, squiggly arrows represent views, and the normal arrows represent named structures. Here LF is used as a logical framework to define some logics, which are then used as meta-theories for algebraic theories. We see three pragmatic levels: the logical frameworks at the top, logics in the middle, and the domain theories at the bottom. Meaning trickles down from the theories at the top (the ones without meta-theories), which are implemented directly in MMT/Scala as described for LF above. This setup can even encode model theory theory morphisms into semantic theories like ZFC set theory.

4 Logical Aspects of the Translation

The logical basis of our export is a definition of Pure in the MMT system. MMT allows defining a wide variety of logical frameworks, and we use PLF as a starting point, a polymorphic variant of LF [15] that already exists in the MMT standard library [37].

4.1 Type System and Logic

Types, Terms, Propositions We use a shallow embedding of Pure in PLF. Besides simplicity, this has a critical scalability advantage: a deep embedding would lead to substantially larger PLF-expressions when already our shallow embedding ended up yielding the largest export size we had ever attempted (since then eclipsed only by our analogous export for Coq [38]). Consequently, as Pure uses shallow polymorphism (type variables bound at the outside of declarations), we cannot use LF itself but need to extend it with shallow polymorphism. That is why we use PLF instead.

Using a shallow embedding, most Pure primitives are represented as their PLF-counterparts: Pure-types and terms are represented as PLF-types and terms. This includes in particular Pure's simple **function** types, λ-abstractions, and application.

The remaining primitives can simply be declared as PLF-constants. That yields a PLF-theory containing in particular the constants

- $\texttt{prop} : \texttt{type}$ for the type of **propositions**,
- $\texttt{ded} : \texttt{prop} \to \texttt{type}$ mapping each proposition φ to the type $\texttt{ded}\,\varphi$ of **proofs** of φ.

That is the bare minimum to connect Isabelle/Pure to PLF: the remaining connectives are produced from the regular export of the `Pure` theory itself, yielding further constants:

- $\mathit{Pure.eq} : \Pi_{a:\texttt{type}}\, a \to a \to \texttt{prop}$ polymorphic **equality** (with implicit $\alpha\beta\eta$-conversion),
- $\mathit{Pure.all} : \Pi_{a:\texttt{type}}\, (a \to \texttt{prop}) \to \texttt{prop}$ for the polymorphic binder for **local parameters**,
- $\mathit{Pure.imp} : \texttt{prop} \to \texttt{prop} \to \texttt{prop}$ for the constructor for **logical entailment**.

Relative to these declarations, it is straightforward to translate all Isabelle types, terms, and propositions.

Proof Terms Like LF but unlike Pure, PLF offers dependent types. These are not needed for representing the simply-typed Pure language but are helpful to concisely represent Pure-proofs as PLF-terms in Curry-Howard style. Thus, Pure proof terms can be exported analogously to types, terms, and propositions. However, in practice, we only export proof terms for small examples because proof terms for actual Isabelle/HOL are far too big. After our work on Isabelle, we conducted a similar export for Coq in [38]. Here we included proof terms, and the sizes, while large, remained manageable. But due to the lack of Coq-style implicit computation, we expect Pure proof terms to be even larger.

However, there is a separate, deeper reason to defer proof exports: it is still unclear what the best way to export proofs is. The export of low-level proof terms is straightforward, but the proof objects are huge and have only limited value (independent proof checking being the main one). The high-level proofs seen by the user are much more interesting but lack the information inferred by the prover.

Therefore, we opted for exporting all proofs as dummy terms that carry only the information that the theorem was checked by Isabelle and which dependencies were used. Additionally, we include, as an informal narrative text, the command-source of the Isar text: this treats the whole proof as one unit, without the hierarchical structure of Isar proofs (see also the discussion in 4.4 and 6.5 below).

4.2 Declarations

Foundational Declarations It is straightforward to represent the foundational declarations of Pure theories as PLF-declarations as follows:

- Pure-**type operators** a of arity n as n-ary PLF-constants

  $a : \texttt{type} \to \ldots \to \texttt{type} \to \texttt{type}$

- Polymorphic Pure-**terms** c of type A using type variables $a_1, \ldots, a_n$ as PLF-constants

  $c : \Pi_{a_1:\texttt{type}} \ldots \Pi_{a_n:\texttt{type}} A$

- Polymorphic Pure-**axioms** s with type parameters $a_1, \ldots, a_n$ asserting proposition F as PLF-constants

  $s : \Pi_{a_1:\texttt{type}} \ldots \Pi_{a_n:\texttt{type}} \texttt{ded}\, F$

All three kinds of declarations may carry definitions, which can be represented by giving the PLF-constant a definiens. This is used only for type operators and term *abbreviations*. HOL type definitions are a special case of high-level declarations as described below, and Pure term definitions are mapped to definition-less constants with defining axioms (multiple ones in case of overloading). Additionally, theorems are represented using the proof as the definiens (as described above).

Identifiers Isabelle assigns to each foundational declaration a unique identifier. It uses separate namespaces for types, terms, and theorems and usually qualifies their names by the base name of the enclosing theory. Every theory exists within an Isabelle session, whose name usually qualifies the theory's base name. Both qualification schemes are optional — there is no strict enforcement.

For reusability, it is preferable to use a single namespace (to ensure globally unique identifiers for all declarations) and to use a uniform naming schema for all identifiers. Moreover, MMT requires all names to be globally unique by qualifying them with an ownership-defining URI. So we have chosen the following naming scheme for all declarations:

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

where *long-theory-name* is the session-qualified theory name, *entity-name* the declaration name within the theory context, and *entity-kind* its name space: notably `type`, `const`, `thm`, or other name spaces of user-defined concepts. For example,

`https://isabelle.in.tum.de?HOL.Nat?Nat.nat|type`

refers to the type `nat` of natural numbers in the theory `Nat` in the session `HOL` of the main Isabelle library. The seemingly redundant repetition of `Nat` is needed to cover corner cases, including some unqualified names in Isabelle/Pure.

High-Level Declarations Isabelle provides a user-extensible set of high-level specification elements, whose semantics is defined by elaboration into foundational ones. Examples include HOL-type definitions or the definition of inductive data types and recursive functions. Similarly, the high-level specification contexts of locales and type-classes (see below) are elaborated into primitive concepts of the logic. Both are already covered by exporting their elaboration, but that results in representations without the high-level structure seen by users.

MMT provides a similar extensible declaration pattern mechanism [16, 39] so that we can use them to represent Isabelle's high-level declarations in a structure-preserving way. We have so far carried out this effort only for locales and leave other elements to future work: it could be done by a generic Isabelle/ML interface for such specification elements such that the export works uniformly for all its instances. Then a manageable separate implementation effort would be needed for each specification element. However, because the individual specification elements were implemented by different authors and can be very complex, no single person could retrofit them to implement this interface, and a long-term community effort is required.

4.3 Module System

Theories The MMT module system subsumes the expressivity of Isabelle theories and is available for every language defined in MMT such as PLF. Thus, all Isabelle theories (including those for logics like HOL) are represented straightforwardly as PLF-theories.

Locales As the Isabelle logical framework lacks primitive support for "little theories", a locale definition is elaborated into a constant definition (predicate) for the logical specification, together with extra-logical management of the resulting context and conclusions produced within it [25]; similar techniques are used for Isabelle type classes [13] on top of locales.

Without any special care, the export of locales merely shows these predicate definitions with theorems depending on additional parameters and premises. But this low-level elaboration is not what Isabelle users users expect. Instead we refer to exported information about the original structure of locale specifications and map that to first-class theories in MMT. Subsequently, we illustrate this approach by a representative example.

Semigroups Consider the following locale for semigroups. It declares (fixes) the binary operation (where we write `x*y` for `op x y`), assumes the associativity axiom, defines the squaring function, and states a simple theorem:

```
locale sg =
  fixes op :: 'a → 'a → 'a  (infixl * 70)
  assumes assoc: ∀ x y z. (x * y) * z = x * (y * z)
```

```
begin
  definition sq :: 'a → 'a where sq x = x * x
  theorem sqsq: sq (sq x) = x * sq x * x <proof>
end
```

Note that the universe of the semigroup is not declared explicitly. Instead, Isabelle locales treat any type variable that remains uninstantiated after type-checking as a type fixed in the locale. In our PLF representation, this convention is made explicit by declaring the universe a as a type and then treating all fixed types and operations uniformly. In the sequel, we use the words *structure* to refer to a tuple of values interpreting the fixed types and operations, and *instance* for a structure that satisfies the assumed axioms.

Translation by Elaboration The locale's elaboration is represented as the following set of PLF-constants (where we again write $x * y$ for $op\, x\, y$ but op is now always a bound variable):

- one membership predicate that ranges over structures and a defining axiom for it that makes it true for instances:

  $sg : \Pi_{a:\mathtt{type}}\, \Pi_{op:a\to a\to a}\, \mathtt{prop}$

  $sg\_def : \Pi_{a:\mathtt{type}}\, \Pi_{op:a\to a\to a}\, \mathtt{ded}\,(sg\, a\, op) \Leftrightarrow \forall x, y, z.(x * y) * z = x * (y * z)$

- for every definition, a global constant and a defining axiom for it, both abstracting over structures:

  $sg.sq : \Pi_{a:\mathtt{type}}\, \Pi_{op:a\to a\to a}\, a \to a$

  $sg.sg\_def : \Pi_{a:\mathtt{type}}\, \Pi_{op:a\to a\to a}\, \mathtt{ded}\, d = \lambda_{x:a}\, x * x$

- for every theorem, a global theorem abstracting over structures and relativized to instances:

  $sg.sqsq : \Pi_{a:\mathtt{type}}\, \Pi_{op:a\to a\to a}\, \mathtt{ded}\,(sg\, a\, op) \Rightarrow \forall x. SQ\,(SQ\, x) = x * (SQ\, x) * x$
  $:=$ (proof omitted)

  (abbreviating $sq.sq\, a\, op$ as SQ).

Note that Isabelle's elaboration introduces the function $sg.sq$ for all structures even though it is only defined for instances. This is sound in the special case of Isabelle because function types are simple and all types are non-empty (which makes adding unspecified operations conservative) and because all locale theorems are relativized to instances.

Reconstruction of Isabelle Locales s MMT Theories By elaborating locales into global declarations, some information about the modular structure is lost. To allow for preserving that structure, we additionally and redundantly export every locale as a PLF-theory with the following local declarations:

- a primitive constant for all fixed types and operations and assumed axioms:

  $a : \mathtt{type}$

  $op : a \to a \to a$

  $assoc : \mathtt{ded}\, \forall x, y, z.(x * y) * z = x * (y * z)$

  (writing $x * y$ for $op\, x\, y$),

- a defined constant for each definition and theorem:

  $sq : a \to a \ := \ \lambda_{x:a}\, x * x$

  $sqsq : \texttt{ded}\, \forall x.sq\,(sq\,x) = x * (sq\,x) * x \ := \ [\text{proof omitted}]$

This nicely conforms to the intention of Isabelle locales as extra-logical add-ons to the Pure logic. We represent sublocale relations and locale interpretations as PLF theory morphisms accordingly (by re-using exported information from Isabelle locale management).

Type Classes Type classes are a special case of locales with some add-on infrastructure, notably for type inference. A locale may become a type class if it has exactly one free type variable `'a`.

If *sg* is instead declared as a type class, the following additional declarations are present:

- for every fixed operation, a global constant abstracting only over the single fixed type:

  $sg\_class.op : \Pi_{a:\texttt{type}}\, a \to a \to a$

- for every assumed axiom, a corresponding global axiom relativized by the membership predicate *sg* of the locale (instantiating the fixed operation *op* with $sg\_class.op\,a$):

  $sg\_class.assoc : \Pi_{a:\texttt{type}}\, \texttt{ded}\, sg\,a\,(sg\_class.op\,a) \Rightarrow \forall x, y, z.(x * y) * z = x * (y * z)$

  (writing $x * y$ for $sg\_class.op\,a\,x\,y$)
- for every definition, a corresponding global constant with a defining axiom,
- for every theorem, a corresponding global theorem.

4.4 Ontology

The description above covers the translation of all logical content. But it is useful to additionally export a high-level abstraction of the library ontology in semantic web style. This includes all named entities (locales, theorems, etc.) and their interrelations but excludes all complex objects (types, terms, proofs).

Such an ontology export is easier to maintain efficiently, e.g., using RDF triple stores. And it is sufficient for many important applications such as querying the dependency relation between declarations. Additionally, it can easily include metadata such as check times.

Isabelle/MMT performs such an RDF/XML export as well, see also 5.3 for the amount of relational information. We originally presented this RDF export in [9] together with an Upper Library Ontology (ULO) that describes and provides a uniform vocabulary of classes and relations for all proof assistants; therefore, we mention only a few recent improvements here. The relational ontology also captures some aspects of inductive and primitive recursive definitions (via the binary relation `ulo:inductive-on`). Most importantly, our export now fully covers dependencies, spanning a large dependency graph over the source text: it relates via the binary relation `ulo:uses` every theorem statement with every used constant and every proofs with every used theorem.

5 Technical Aspects of the Translation

The majority of the export is not OMDoc-specific and carried out on the Isabelle side; this appeared first in the official release Isabelle2019 (June 2019), but the present paper uses the reworked and simplified version of Isabelle2020 (April 2020). Being integrated into

Isabelle has the advantage that most of our work can be immediately reused for exports into other formats than OMDoc. Only little OMDoc-specific code is necessary for building and serializing the XML objects in OMDoc format. For this, we use the Mmt API for OMDoc, which is also written in Scala and therefore directly callable from PIDE. This code is now part of the Mmt distribution (first in release 14 from November 2018).

The resulting inter-dependency between the code bases is handled as follows: if the Mmt directory is registered to Isabelle as *component*, it provides a tool `isabelle mmt_build` (shell script) to build MMT with Isabelle support enabled. The resulting `mmt.jar` will provide further tools `isabelle mmt_import` and `isabelle mmt_server` (in Scala) to perform the import and view its results. Users merely need to invoke, e.g., `isabelle mmt_import -B ZF`.

5.1 Export from Isabelle

Isabelle/Scala provides a standard module `Export_Theory` to expose theory content to other tools via a statically typed API that imitates Isabelle/ML datatypes for types and terms. The communication between Isabelle/ML and Isabelle/Scala works via untyped XML trees, without any special tricks about meta-programming. Instead, sources in both languages reside next to each other in the official Isabelle repository, are manually updated accordingly.

A first version of the Isabelle export facility appeared in Isabelle2018 (August 2018). It was originally motivated by early versions of Isabelle/MMT, and has grown into an independent Isabelle service. It is supported by command-line tools like `isabelle export` and `isabelle dump` [53]; `isabelle build` with option `export_theory` exposes logical content as follows.

- Foundational theory content of the Isabelle/Pure *logical framework*: **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). Actual proofs are not exported by default — they are prohibitively large. The option `export_standard_proofs` provides proof terms in a standardized format that facilitates import in other tools, but this only works for small examples so far.
- Constant definitions of Isabelle/Pure, as a relation between a single constant with multiple axioms. Overloading in Isabelle means that a polymorphic entity is characterized on multiple (non-overlapping) type instances. The majority of constants are non-overloaded, with exactly one equational axiom to express its definition. This relation of constants to their defining axioms is exported, too.
- Type definitions of Isabelle/HOL in the sense of Gordon and Pitts [45]. This axiomatization scheme can be interpreted definitionally within the standard semantics of the HOL logic. Isabelle/HOL provides a separate module to create new types via that mechanism. Some key information is exported: the old representing type, the new abstract type, the name of the morphisms between the two with the axiom stating the relation.
  This allows recovering HOL typedefs faithfully, where 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. This works by querying generic information in Isabelle/Pure about functional or relation specifications (also known as "Spec Rules"). The Isabelle/HOL implementations provide this data on their own account.
  This merely provides a rough classification of term constants at a very abstract level. The full complexity of Isabelle/HOL specification mechanisms is more difficult to capture:

it would mean to follow many implementation details, including ones that have changed fundamentally over the years of ongoing Isabelle development.
- Dependencies of proven theorems wrt. types, consts, theorems, as recorded by the Isabelle inference kernel: This spans a large dependency graph over the document in terms of the primitive logic — extra-logical aspects are missing (e.g., dependency on notation). Partial support for these *proof constants* had been part of the Isabelle codebase over many years, but we had to rework this substantially to make it suitable for our application.
- Locales in the sense of Ballarin [3] and type classes as special locale interpretations in the sense of Haftmann and Wenzel [13, 14]: 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). These are then exported as Mmt theory morphisms. For type classes, the export shows the canonical locale interpretation but without an explicit connection to the type class. This would have to be a type-indexed family of Mmt theory morphisms.
- 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 [40]: This allows reconstructing Isabelle's built-in type class reasoning by an external program (for example, an application could give it to a separate process running Isabelle/Pure and reuse the original implementation in module `Sorts` Isabelle/ML). An alternative is to imitate these operations in a different programming language.[2]

Formal entities have two name components: *kind* (to distinguish the namespace) and *full name* (usually with the theory base name as qualifier). In addition, there is an *external name* for printing (partially qualified according to standard namespace policies), a *source position*, and a *command span identifier*. The latter allows in particular arranging the content according to the order in which it occurs in the source text so that exported types, constants, theorems appear as a digest for each specification element in the text (e.g. for **definition**).

Moreover, if the target format of the export supports references to the original source, this can be used to attach such a reference or even the entire source fragment to each formal entity. We do that for our OMDoc export.

5.2 Import into Mmt

The entities listed in Section 5.1 can be serialized almost directly as Mmt constants relative to the PLF framework as described in Section 4. That is not surprising as much of that work motivated by the present export in the first place. Figure 3 shows the Mmt browser displaying an example that is very small and thus includes proof terms. Note how every formal declaration is preceded with an informal narrative fragment containing the original source text, this is for the orientation for Isabelle users.

In the sequel, we describe a few specific adaptations of the term language that were required to reconcile traditional Isabelle/ML representations with the more conventional λ-calculus of PLF in Mmt.

Type arguments for constants The traditional representation of polymorphic constants in Isabelle and the HOL family [45] is to give the full *type instance* at each occurrence in a term, instead of the *type arguments* that produce the instantiation of the general type schema. For

[2] Isabelle/Scala does not provide any type-class reasoning on its own, because it is meant to be for external system management only. Logical operations are done properly in Isabelle/ML.

example, constant `id :: 'a => 'a` occurs in particular terms as the pair `(id, τ => τ)` for the respective type τ. This is both redundant (because the type instances are usually bigger than the type arguments) and inconvenient (because it is more difficult to obtain the type arguments from the instantiations than the other way around). In contrast, PLF treats `id` as a function with dependent type $\Pi_{a:\texttt{type}}\, a \to a$ and occurrences are just applications $(id\ \tau)$.

Isabelle/ML provides operations to switch between the two representations within a given context of constant declarations. Our theory export always uses the second form with type arguments: this reduces the size of exported material and allows importing terms into PLF without again referring to the environment of constant declarations.

Variable names Isabelle variables come in various flavors: free variables (e.g., `x`), schematic variables with index (e.g., `?x10`), and bound variables (e.g., `x` in λ`x::`τ`. x`) which is notation for the de-Bruijn index abstraction `Abs (x, τ, B.0)` where `x` is retained as a comment).

To fit smoothly into the λ-calculus of PLF, 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 statement. And bound variable comments in abstractions are renamed locally to avoid clashes with free variables in the same scope. Thus the `Abs` comment can be used literally in PLF as a named abstraction ignoring the unnamed de-Bruijn index representation of Isabelle.

Type class constraints 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 links certain operations to overloaded term constants (e.g., `less :: 'a => 'a => bool`) and ensures logical premises on these operations (e.g., stating that `less` is a strict order on the type).

Isabelle type class operations are managed by extra-logical means to eliminate the implicit overloading. In PLF this merely results in multiple constant definitions for different type arguments. Class premises become logical constraints in a straight-forward manner: a type class is a predicate over types in PLF. So `'a::c` means that the predicate `c` applied to type `'a` holds. Statements with class constraints φ(`'a::c`) are augmented by a prefix of preconditions `'a::c` $\Longrightarrow \varphi$(`'a`), effectively eliminating the constraint within the logic.

5.3 Statistics for Isabelle/AFP

Our test hardware for the Mmt export of Isabelle/AFP is a server machine with 40 CPU cores (80 hardware threads), 128 GB RAM (2 NUMA nodes), and fast SSD storage. Below, we give an overview of the material for Isabelle2020 (April 2020) with MMT/52adb5e338811e [20] and AFP/91f1cdbeefc0 [1]: These sources consist of 680 sessions distributed over 7,027 files comprising 160 MB of theory text (30 MB XZ-compressed). The exported content comprises

- 7,027 theories and 5,291 locales ("little theories"), including 1,236 type classes,
- 2,116,638 individuals (11,724 `type`, 204,404 `const`, 236,186 `axiom`, 1,497,689 `thm`).
- 400,996,957 relations, including 386,325,246 `ulo:uses` (i.e. the overall dependency graph of `type`, `const` and `thm` items)
- 65 GB OMDoc/XML (310 MB XZ-compressed)[3]

[3] `https://gl.mathhub.info/Isabelle/Distribution/commit/db1009a326c8` and `https://gl.mathhub.info/Isabelle/AFP/commit/346f28873c9f`

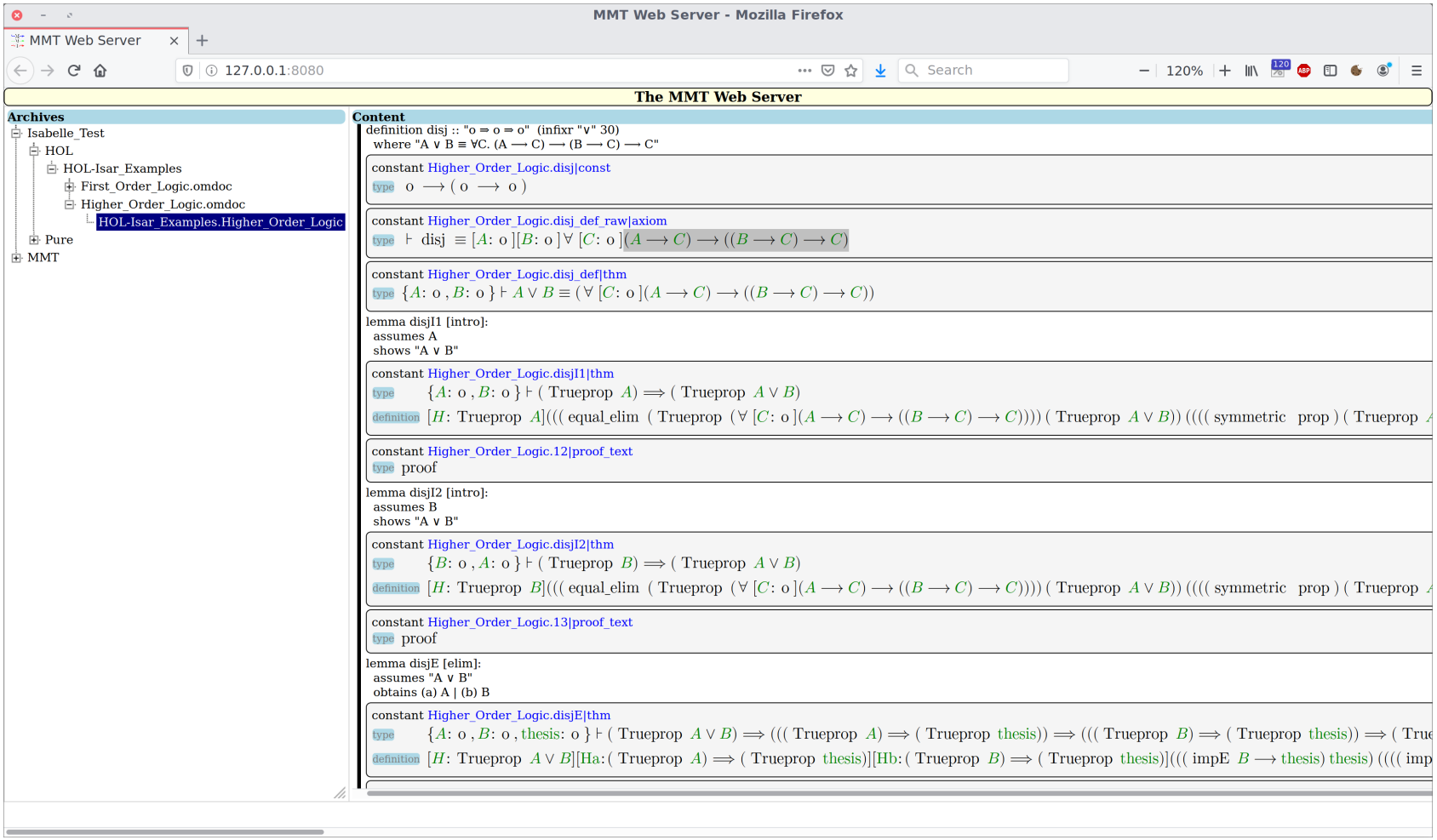

Figure 3 Disjunction in Higher-Order Logic: definitions, theorems and proof terms

The entire process of Isabelle/PIDE document checking, export to MMT, and serialization as XZ-compressed XML requires 80 GB RAM, 8 CPU cores, and 22h30 elapsed time. Thus, compared to an elementary batch-build, our export requires around 2 times the memory and 2–5 times the elapsed time (mainly because Isabelle/MMT uses less parallelization than `isabelle build`). We emphasize that these resource figures are for the *entire* AFP, including the special sessions tagged as `slow` or `large` which are often omitted because they take a lot of resources to process.

The size of the exported OMDoc data structures is linear in the size of the original sources, increased by about factor 10 in XZ-compressed form. This increase in size is a gain, not a deficiency — it stems from the fact that the exported XML contains substantial additional information that is implicit in the sources but extremely difficult to infer: all occurrences of symbols are disambiguated and exported with their unique URIs; the exported XML elements carry source references, i.e., URIs that link to the corresponding location in the source; all type arguments of occurrences of polymorphic constants and all types of bound variables are included in the XML even if omitted in the sources; and all theorems automatically generated by Isabelle are included in the export. We could suppress some of this information, but that would defeat the purpose of our export: only Isabelle can infer all details, and handing it to other tools is our export's main value. The uncompressed XML files are much larger because they are very verbose and optimized for context-free processing. But we never write the XML directly to the file-system: all reading and writing of XML is filtered through XZ compression.

5.4 Maintainability

When developing proof assistant library exports, the challenge of maintainability is often overlooked or underestimated. This is partly caused by the incentives of the academic system that rewards quickly published results rather than long-term sustainable ones. We have consciously taken several steps to ensure maintainability.

Firstly, we use statically-typed Scala APIs as much as possible, both in the export from Isabelle and in the import into MMT. Almost all the new code we wrote for the occasion was immediately integrated with the existing abstract interfaces. The remaining glue code that connects Isabelle's abstract export with MMT's abstract import comprises only a few thousand straightforward lines of code.

Secondly, wherever possible we wrote new code in the Isabelle repository rather than the MMT repository. This forces future Isabelle development to maintain our abstract code, in particular when PIDE data structures change. Concretely, we pushed only the parts of the code that actually depend on the MMT data structures to the MMT repository. That portion consists of only about 2000 lines of code, mostly straightforward code for creating instances of the MMT data structures. The rest of the export code is generally reusable for other Isabelle exports and pushed to the Isabelle repository and already released as an official Isabelle feature. In fact, this design has already proved beneficial as Wenzel was able to reuse the Isabelle part of our code in a recent export to Dedukti (still unpublished).

Finally, the fact that Isabelle and MMT can communicate via the Java VM has proved a huge advantage for maintainability. We were able to design the code in such a way that MMT is an optional plugin component for Isabelle and vice versa. Thus, users running Isabelle can simply register MMT as a plugin with Isabelle and then run `isabelle mmt_import` on the command-line.

Whenever a new Isabelle release is published, it will be a matter to update some statically-typed Scala functions for Isabelle/MMT. Informed by our experience of multiple similar

exports, we judge this one to be the most maintainable export of a proof assistant library so far, in fact by a wide margin.

6 Enabled Applications

Our work now allows exporting entire Isabelle libraries into a format that can be easily read by third-party applications in a robustly maintainable way. A major motivation for this work was enabling applications that use this exported data. However, it remains open which applications should be better realized directly in Isabelle and which should be based on MMT. Critically, our export abstracts from most idiosyncrasies of Isabelle's logic, implementation, and library structure. That has advantages and disadvantages.

On the positive side, any application that does not significantly depend on Isabelle's code base (e.g., search or dependency management) or explicitly rejects using it (e.g., representations in a logical framework or external proof checking) benefits from the uniform representation in the relatively simple language of MMT. On the negative side, any application that should be tightly integrated with Isabelle may be better realized natively in Isabelle. This includes in particular applications that offer proof advice or rewrites/generates Isabelle data structures or Isabelle sources.

In some cases combined approaches may be indicated such as a small native addition to Isabelle that connects to a service implemented on top of the MMT representation (and possibly running on a high-performance remote server). For example, search services could be realized well in this way. However, even when a native implementation that ignores the import into MMT is indicated, our work can provide substantial benefits. Any such native implementation will likely benefit from our streamlining and scaling up of Isabelle's export capabilities that allow integrating such applications with Isabelle.

Ultimately, the assessment which of these effects dominate must be made on a case-by-case basis for every application. In the sequel, we sketch some applications enabled by our work where we expect the advantages to dominate.

6.1 Clarification of Isabelle/Pure in Terms of MMT/PLF

The Isabelle/Pure framework [43] is historically connected to Edinburgh LF, but it has its own distinctive style that can obscure important aspects. The documentation [51, §2] refers to related formulations of λHOL within the setting of Pure Type Systems (PTS) due to Barendregt and Geuvers [4] and gives informal explanations (in LaTeX) about how to understand Isabelle-specific concepts like schematic variables or type-classes.

Instead of Isabelle folklore and informal explanations in the documentation, our translation to PLF within MMT elucidates many concepts of Pure more formally. In particular:

- The three levels of λ-calculus for function spaces (higher-order abstract syntax), universal binding of local parameters (quantification), logical entailment of rule statements (implication) become just one dependently-typed λ-calculus.
- Implicit polymorphism becomes explicit as abstraction and quantification over types.
- Up to scalability issues, proof terms — which are an optional add-on to the Pure logic — become plain λ-terms as definiens for theorems.
- Type class constraints become explicit as predicates applied to types. Concretely, there are two possible representations for extra-logical constraints: `'a::c` and intra-logical predication `OFCLASS('a, c_class)`. Both are turned into the obvious term `c a` for `c :: type => prop` in PLF).

Still lacking in our export is the explicit treatment of *type class parameters*: as in Isabelle/Pure, the PLF theory treats instance-specific definitions as a collection of axioms that are associated with a generically typed constant. A more sophisticated translation could try to make a dictionary construction, to turn type class parameters into explicit function parameters everywhere.

6.2 External Proof Checking

An often asked-for application of an Isabelle export is independent re-verification. It may appear straightforward to use our export as the input of a separate application that specializes on re-checking proofs. However, while this is certainly one of the intended uses, it would be naive to assume that our work is more than the first of multiple steps towards this goal. In the sequel, we describe the remaining two obstacles: scalability and adequacy. These obstacles are not inherent to our approach. We expect any future solution to external proof checking to build on our approach or to recreate something comparable.

Regarding **scalability**, it is indeed straightforward to write a proof-checker for the Pure logic underlying Isabelle. In fact, the Mmt formalization of Pure induces a proof-checker for Isabelle out of the box. Similar framework-induced checkers can be built easily in implementations of LF-like frameworks such as Dedukti. Moreover, the complexity of these checkers would typically be linear in the size of the proofs and thus very feasible. It is even possible that checking the proofs could be faster than the file-system access needed to read the proofs in the first place.

But we do not expect such straightforward checkers to be able to handle the size of the proofs in the library: the size of individual proofs, if naively encoded, may very well exceed the memory capacity of typical checkers.[4] Thus, additional investments are needed for handling large proofs, such as structure sharing, inferring omitted trivial steps, or streamed processing that can check a proof without loading it in its entirety. These technologies are known in principle, but applying them to Isabelle/AFP remains substantial future work.

Regarding **adequacy**, note that our export is foundational in the sense that it exports the representation relative to the Pure logic in Isabelle's kernel, which arises from the original user input through a series of highly non-trivial transformations (elaboration). Fully re-checking the proofs that result from elaboration is only one of two necessary conditions. The other one is *conservativity* of elaboration, i.e., the requirement that elaboration does not translate an unprovable statement to a provable one. Depending on how many advanced Isabelle features are used in a problem statement, trusting the conservativity of elaboration may be a bigger leap than trusting the correctness of the proofs.

But conservativity is extremely difficult to establish. The most direct way would be to specify the semantics of Isabelle's surface syntax and then prove Isabelle's elaboration algorithms correct relative to it. Given the complexity of elaboration, this remains out of reach in the foreseeable future.

6.3 Dependency Management

The classic model of Isabelle/PIDE [49] document markup merely provides a record of formal entities that are *explicitly visible* in the source text. Due to some reworking of the inference kernel by Wenzel, there is now a detailed record of all `type` / `const` / `thm` entities that are

[4] Early experiments conducted with parts of the Main theory context of Isabelle/HOL produce hundreds of megabytes of proof terms in textual representation.

implicitly used. This spans a rather large dependency graph over the original source: for Isabelle/AFP there are 400 million edges for 130 MB of theory text.

In the past, users have occasionally attempted to approximate this information for their own purposes, e.g. in the Levity tool [8], which exploits dependencies to move lemmas to adequate locations in the theory hierarchy.

Our ontological export (see Section 4.4) now includes a detailed record of both explicit source dependencies and implicit logical dependencies. With this information available in a standard format, more ambitious (and more robust) refactoring tools can be realized for Isabelle. Optionally, such refactoring tools can even be built in OMDoc/Mmt to work uniformly for all systems that have exports similar to the one reported in this article.

6.4 Search

Because our export includes all logical information of the Isabelle content, it enables multiple search applications. For example, this would allow searching for expressions or names that are not explicitly part of the sources and only occur in inferred information. It also enables applying generic search systems to the Isabelle libraries.

As an example, we sketch a unification-based search service for the entire AFP based on MathWebSearch [33]. MathWebSearch maintains a substitution tree index that allows efficient unification queries over large collections of terms. Because it can index Mmt terms, it can be directly applied to our export. Thus, users can explore the full background library without having it loaded into the prover process (which might require too much memory), or even without installing the prover at all (e.g., by using a web service for the AFP).

Concretely, the queries would be terms with free variables over some AFP theory, and the search results would be terms in the AFP that unify with the query. Because our export includes source references for all entities, these results can be linked to other resources (e.g., the location in the official AFP web site) or directly imported into PIDE.

The main remaining technical hurdle is the processing of the user's query. In order to match anything in the library, formal objects in the query must be processed and exported in the same way as the library. This includes the use of special forms for pattern matching, lists enumeration and comprehension etc. as well as type inference and type matching (with type classes). Moreover, the user must provide the right context in which to interpret the query.

An intermediate solution could run a prover session of reasonable size that contains the most relevant notation (e.g., `HOL-Analysis`) and process queries relative to it. These queries could then be exported and matched against the entire AFP.

We estimate that such a system is within reach of an ambitious Master's thesis.

6.5 Enabling Cross-Library Knowledge Management

Isabelle/MMT is one of multiple large exports of proof assistant libraries that we have conducted over the last few years. One of the original motivations of these efforts was to obtain multiple libraries in a uniform format in order to then develop develop cross-library and cross-prover knowledge management solutions.

These efforts are still at an experimental stage, and we only cite a few early results that could be extended to the Isabelle export:

- We have used alignments [21] to relate corresponding concepts in different libraries. These can be annotated manually or found by machine learning techniques [10]. Given a sufficient alignment coverage, we can then translate terms between libraries and use this to make systems interoperable.

- With the relational RDF/XML export of Section 4.4, we can use SPARQL queries using the Upper Library Ontology (see [9] for details) that return results from multiple libraries. [5] presents an architecture for multi-aspect search based on these ideas.
- In [36] we have presented first steps towards finding views between different theorem prover libraries automatically.

7 Conclusion and Future Work

Summary In this article, we report on the conclusion of a research objective that seemed quite immediate two decades ago, but was not: the export of a theorem prover library (Isabelle) into a FAIR [55] knowledge exchange format (OMDoc). To make this undertaking feasible at all, both the source and target system had to evolve considerably: Isabelle had to add its Scala and PIDE infrastructure to manage and expose document-oriented information in an instrumentable way, and the OMDoc format had to be re-engineered, extended, and implemented in the Mmt system. Of course, the growth of the Isabelle library during this time induced further scalability problems, which we had to solve for our export.

Exports of theorem prover libraries have received substantial attention for the last 10–20 years. Our work is the first comprehensive export for Isabelle: we demonstrate current Isabelle/Scala export technology and explore remaining theoretical and practical challenges.

Even ignoring the potential applications of this particular export, our infrastructure for exporting Isabelle libraries in general will prove beneficial to future improvements to Isabelle itself and to the reuse of Isabelle content in other systems. In fact, the improvements of Isabelle that were needed for our export have already shown benefits for the wider Isabelle community. The *headless PIDE* session and `isabelle dump` tool have become particularly important: we are in personal contact with two different projects to build content-oriented search engines on top of these systems. Another emerging application of this technology is a similar export of Isabelle to Dedukti [7]: this aims at re-checking the Isabelle/AFP and therefore includes proof terms but excludes PIDE document markup.

The current export facility is mostly based on code that is maintained within the Isabelle repository, and thus updated by the core developers. We have already published Isabelle/Mmt for Isabelle2019 and Isabelle2020 based on a straight-forward process that users can easily recreate themselves: build Mmt within the Isabelle system environment, turn it into an Isabelle component, and use the standard Isabelle release tool to build a stand-alone variant of Isabelle that includes Mmt. Users can then rerun our export themselves on the spot (via the `isabelle mmt_import` command). We judge that this makes our Isabelle export the most easily reproducible and maintainable among all existing prover library exports.

Future Work Besides realizing and scaling up the applications described in Section 6, we want to mention two important avenues for future work:

- The current export does not include proof objects as these would increase its size by an order of magnitude. Instead, we restrict ourselves to the dependency relation induced by the proofs, which already enables many applications, but not, e.g., re-verification of proofs. To obtain scalable proof exports, we must investigate how to shrink the size of the proofs, e.g., by developing a new language for high-level proofs.
- In a similar vein we want to preserve the structure of more high-level declarations — e.g. HOL-type definitions, inductive types. As discussed in Section 4.2, this is supported by Mmt and would allow a structurally more similar and thus more understandable export.

References

1 Archive of formal proofs (AFP), April 2020. URL: https://www.isa-afp.org.

2 S. Autexier, D. Hutter, H. Mantel, and A. Schairer. Towards an Evolutionary Formal Software-Development Using CASL. In D. Bert, C. Choppy, and P. Mosses, editors, *WADT*, volume 1827 of *Lecture Notes in Computer Science*, pages 73–88. Springer, 1999.

3 Clemens Ballarin. Locales: A module system for mathematical theories. *JAR*, 52(2):123–153, 2014. URL: https://doi.org/10.1007/s10817-013-9284-7.

4 H. Barendregt and H. Geuvers. Proof assistants using dependent type systems. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, 2001.

5 Katja Berčič, Michael Kohlhase, and Florian Rabe. Towards a heterogeneous query language for mathematical knowledge – extended report, 2020. URL: http://kwarc.info/kohlhase/papers/tetrasearch.pdf.

6 J. Betzendahl and M. Kohlhase. Translating the imps theory library to mmt/omdoc. In F. Rabe, W. Farmer, G. Passmore, and A. Youssef, editors, *Intelligent Computer Mathematics*, volume 11006, pages 7–22. Springer, 2018.

7 M. Boespflug, Q. Carbonneaux, and O. Hermant. The λΠ-calculus modulo as a universal proof language. In D. Pichardie and T. Weber, editors, *Proceedings of PxTP2012: Proof Exchange for Theorem Proving*, pages 28–43, 2012.

8 Timothy Bourke, Matthias Daum, Gerwin Klein, and Rafal Kolanski. Challenges and experiences in managing large-scale proofs. In Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *Intelligent Computer Mathematics - 11th International Conference, AISC 2012, 19th Symposium, Calculemus 2012, 5th International Workshop, DML 2012, 11th International Conference, MKM 2012, Systems and Projects, Held as Part of CICM 2012, Bremen, Germany, July 8-13, 2012. Proceedings*, volume 7362 of *Lecture Notes in Computer Science*, pages 32–48. Springer, 2012. doi:10.1007/978-3-642-31374-5_3.

9 A. Condoluci, M. Kohlhase, D. Müller, F. Rabe, C. Sacerdoti Coen, and M. Wenzel. Relational Data Across Mathematical Libraries. In C. Kaliszyk, E. Brady, A. Kohlhase, and C. Sacerdoti Coen, editors, *Intelligent Computer Mathematics*, pages 61–76. Springer, 2019.

10 T. Gauthier and C. Kaliszyk. Matching concepts across HOL libraries. In S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, editors, *Intelligent Computer Mathematics*, pages 267–281. Springer, 2014.

11 M. J. C. Gordon. HOL: A machine oriented formulation of higher order logic. Technical Report 68, University of Cambridge Computer Laboratory, 1985.

12 M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *LNCS*. Springer, 1979.

13 F. Haftmann and M. Wenzel. Constructive Type Classes in Isabelle. In T. Altenkirch and C. McBride, editors, *TYPES conference*, pages 160–174. Springer, 2006.

14 Florian Haftmann and Makarius Wenzel. Local theory specifications in Isabelle/Isar. In Stefano Berardi, Ferruccio Damiani, and Ugo de Liguoro, editors, *Types for Proofs and Programs, TYPES 2008*, volume 5497 of *LNCS*. Springer, 2009.

15 R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

16 Fulya Horozal, Michael Kohlhase, and Florian Rabe. Extending MKM formats at the statement level. In Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *Intelligent Computer Mathematics*, number 7362 in LNAI, pages 65–80. Springer Verlag, 2012. URL: http://kwarc.info/kohlhase/papers/mkm12-p2s.pdf.

17 Mihnea Iancu, Michael Kohlhase, Florian Rabe, and Josef Urban. The Mizar Mathematical Library in OMDoc: Translation and applications. *Journal of Automated Reasoning*, 50(2):191–202, 2013. URL: https://uniformal.github.io/doc/applications/LATIN/docs/MizarOMDocAppl.pdf, doi:10.1007/s10817-012-9271-4.

18 Fabian Immler, Jonas Rädle, and Makarius Wenzel. Virtualization of HOL4 in Isabelle. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. URL: `http://drops.dagstuhl.de/opus/volltexte/2019/11076/pdf/LIPIcs-ITP-2019-21.pdf`.

19 Isabelle website, April 2020. URL: `https://isabelle.in.tum.de/website-Isabelle2020`.

20 Isabelle/MMT for Isabelle2020, April 2020. URL: `https://files.sketis.net/Isabelle_MMT-20200421`.

21 C. Kaliszyk, M. Kohlhase, D. Müller, and F. Rabe. A Standard for Aligning Mathematical Concepts. In A. Kohlhase, M. Kohlhase, P. Libbrecht, B. Miller, F. Tompa, A. Naummowicz, W. Neuper, P. Quaresma, and M. Suda, editors, *Work in Progress at CICM 2016*, pages 229–244. CEUR-WS.org, 2016.

22 C. Kaliszyk and J. Urban. HOL(y)hammer: Online ATP service for HOL light. *Mathematics in Computer Science*, 9(1):5–22, 2015.

23 Cezary Kaliszyk and Alexander Krauss. Scalable LCF-style proof translation. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 51–66, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

24 Cezary Kaliszyk and Florian Rabe. Towards knowledge management for HOL Light. In Stephan Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics 2014*, number 8543 in LNCS, pages 357–372. Springer, 2014. URL: `http://kwarc.info/frabe/Research/KR_hollight_14.pdf`.

25 F. Kammüller, M. Wenzel, and L. Paulson. Locales – a Sectioning Concept for Isabelle. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics*, pages 149–166. Springer, 1999.

26 C. Keller and B. Werner. Importing HOL Light into Coq. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving*, pages 307–322. Springer, 2010.

27 M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer, 2006.

28 M. Kohlhase, D. Müller, S. Owre, and F. Rabe. Making PVS Accessible to Generic Services by Interpretation in a Universal Format. In M. Ayala-Rincon and C. Munoz, editors, *Interactive Theorem Proving*, pages 319–335. Springer, 2017.

29 M. Kohlhase and F. Rabe. QED Reloaded: Towards a Pluralistic Formal Library of Mathematical Knowledge. *Journal of Formalized Reasoning*, 9(1):201–234, 2016.

30 M. Kohlhase and I. Şucan. A Search Engine for Mathematical Formulae. In T. Ida, J. Calmet, and D. Wang, editors, *Artificial Intelligence and Symbolic Computation*, pages 241–253. Springer, 2006.

31 Michael Kohlhase. The flexiformalist manifesto. In Andrei Voronkov, Viorel Negru, Tetsuo Ida, Tudor Jebelean, Dana Petcu, Stephen M. Watt, and Daniela Zaharie, editors, *14th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*, pages 30–36, Timisoara, Romania, 2013. IEEE Press. URL: `http://kwarc.info/kohlhase/papers/synasc13.pdf`.

32 Michael Kohlhase and Florian Rabe. Experiences from exporting major proof assistant libraries. submitted, 2020. URL: `https://kwarc.info/people/frabe/Research/KR_oafexp_20.pdf`.

33 Michael Kohlhase and Ioan Şucan. A search engine for mathematical formulae. In Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI, pages 241–253. Springer Verlag, 2006. URL: `http://kwarc.info/kohlhase/papers/aisc06.pdf`.

34 A. Krauss and A. Schropp. A Mechanized Translation from Higher-Order Logic to Set Theory. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving*, pages 323–338. Springer, 2010.

35 J. Meng and L. Paulson. Translating Higher-Order Clauses to First-Order Clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008.

36 D. Müller, M. Kohlhase, and F. Rabe. Automatically Finding Theory Morphisms for Knowledge Management. In F. Rabe, W. Farmer, G. Passmore, and A. Youssef, editors, *Intelligent Computer Mathematics*, pages 209–224. Springer, 2018.

37 D. Müller and F. Rabe. Rapid Prototyping Formal Systems in MMT: Case Studies. In D. Miller and I. Scagnetto, editors, *Logical Frameworks and Meta-languages: Theory and Practice*, pages 40–54, 2019.

38 D. Müller, F. Rabe, and C. Sacerdoti Coen. The Coq Library as a Theory Graph. In C. Kaliszyk, E. Brady, A. Kohlhase, and C. Sacerdoti Coen, editors, *Intelligent Computer Mathematics*, pages 171–186. Springer, 2019.

39 Dennis Müller, Florian Rabe, Colin Rothgang, and Michael Kohlhase. Representing structural language features in formal meta-languages. submitted, 2020. URL: `http://kwarc.info/kohlhase/submit/cicm20-features.pdf`.

40 T. Nipkow and C. Prehofer. Type checking type classes. In *ACM Symp. Principles of Programming Languages*, 1993.

41 S. Obua and S. Skalberg. Importing HOL into Isabelle/HOL. In N. Shankar and U. Furbach, editors, *Automated Reasoning*, volume 4130. Springer, 2006.

42 Karl Palmskog, Ahmet Çelik, and Milos Gligoric. piCoq: parallel regression proving for large-scale verification projects. In Frank Tip and Eric Bodden, editors, *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*, pages 344–355. ACM, 2018. `doi:10.1145/3213846.3213877`.

43 Lawrence C. Paulson. Isabelle: The next 700 theorem provers. In *Logic and Computer Science*, pages 361–386. Academic Press, 1990. URL: `http://www.cl.cam.ac.uk/Research/Reports/TR143-lcp-experience.dvi.gz`.

44 Lawrence C. Paulson, Tobias Nipkow, and Makarius Wenzel. From LCF to Isabelle/HOL. *Formal Aspects of Computing*, September 2019. Springer, London. URL: `https://doi.org/10.1007/s00165-019-00492-1`.

45 A. Pitts. The HOL logic. In M. J. C. Gordon and T. F. Melham, editors, *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, pages 191–232. Cambridge University Press, 1993.

46 F. Rabe. A Logic-Independent IDE. In C. Benzmüller and B. Woltzenlogel Paleo, editors, *Workshop on User Interfaces for Theorem Provers*, pages 48–60. Elsevier, 2014.

47 F. Rabe. How to Identify, Translate, and Combine Logics? *Journal of Logic and Computation*, 27(6):1753–1798, 2017.

48 F. Rabe and M. Kohlhase. A Scalable Module System. *Information and Computation*, 230(1):1–54, 2013.

49 Makarius Wenzel. Asynchronous user interaction and tool integration in Isabelle/PIDE. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving (ITP 2014)*, volume 8558 of *LNCS*. Springer, 2014.

50 Makarius Wenzel. Interaction with formal mathematical documents in Isabelle/PIDE. In Cezary Kaliszyk, Edwin Brady, Andrea Kohlhase, and Claudio Sacerdoti Coen, editors, *Intelligent Computer Mathematics (CICM 2019)*, volume 11617 of *LNAI*. Springer, 2019. `https://arxiv.org/abs/1905.01735`.

51 Makarius Wenzel. *The Isabelle/Isar Implementation*, April 2020. URL: `https://isabelle.in.tum.de/website-Isabelle2020/dist/doc/implementation.pdf`.

52 Makarius Wenzel. *Isabelle/jEdit*, April 2020. URL: `https://isabelle.in.tum.de/website-Isabelle2020/dist/doc/jedit.pdf`.

53 Makarius Wenzel. *The Isabelle System manual*, April 2020. URL: `https://isabelle.in.tum.de/website-Isabelle2020/dist/doc/system.pdf`.

54 Markus Wenzel. Isar — a generic interpretative approach to readable formal proof documents. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics: TPHOLs '99*, volume 1690 of *LNCS*. Springer, 1999.

55 Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3, 2016. URL: `https://doi.org/10.1038/sdata.2016.18`.