Translating the IMPS theory library to MMT/ OMDoc

Jonas Betzendahl and Michael Kohlhase

Computer Science, FAU Erlangen-Nürnberg

Abstract. The IMPS system by Farmer, Guttman and Thayer was an influential automated reasoning system, pioneering mechanisations of features like theory morphisms, partial functions with subsorts, and the little theories approach to the axiomatic method. It comes with a large library of formalised mathematical knowledge covering a broad spectrum of different fields. Since IMPS is no longer under development, this library is in danger of being lost. In its present form, it is also not compatible for use with any other mathematical system.
To remedy that, we formalise the logic of IMPS (LUTINS), and draw on both the original theory library source files as well as the internal data structures of the system to generate a representation in a modern knowledge management format. Using this approach, we translate the library to OMDoc/MMT and verify the result using type-checking in the MMT system against our implementation of LUTINS.

1 Introduction

There are many libraries of formal knowledge, but unfortunately, each one needs a different mathematical software system to interpret it – the system it was originally written for. This barrier of non-interoperability severely limits possible uses of any one mathematical software system as well as progress on that system itself. Helpful tooling that was implemented for one system often can not be used for developing another. Some translations between different systems exist but they are often ad-hoc, only one-directional (although both systems are in frequent use) or overly restricted by the logical frameworks or foundations. Approaching the problem with only direct system-to-system translations without common ground also means inviting scaling problems: for n systems we would need $\mathcal{O}(n^2)$ translations.

Most theorem proving systems today tend to fix (in what we will refer to as the *big theories approach*) one particular logical foundation along with its primitives (i.e. types, axioms, rules, ...) and only use conservative extensions to model domain knowledge (i.e. theorems, definitions, ...). This goes against the way that modern mathematics is usually done on chalkboard or paper, where the foundation is often hidden and almost never directly referred to. It also makes it harder for two systems with different logical frameworks to successfully interact.

Another danger to effective use of mathematical knowledge libraries across software systems is that if one of these systems eventually falls out of use, the

library is in danger of being lost to bitrot as fewer and fewer machines are actually capable of running the required software to interpret it.

In this paper we attempt to "rescue" one library in particular from this fate and make it interoperable at the same time. IMPS is an *I*nteractive *M*athematical *P*roof *S*ystem, originally developed at The MITRE Corporation by William M. Farmer, Joshua Guttman, and Javier Thayer. Its library is home to a large amount of well-developed formalised mathematics with over 180 different theories and over 1200 distinct theorems and their proofs.

The IMPS system itself [TMI] has not been in active development or regular use for well over 20 years now and thus the library is in acute danger of being lost. The system (and its library) is especially interesting because it was the first theorem proving assistant to make heavy use of *theory morphisms* and with emphasis on the *little theories* approach to mathematics.

Concretely, we present a translation of the IMPS library into OMDoc/MMT [Koh06] a content markup scheme for (collections of) mathematical documents (including articles, textbooks, and theorem prover libraries) that shares key design choices with the IMPS system, such as the focus on theory morphisms and adherence to the little theories approach.

Contribution We build on and complete the earlier and incomplete efforts by [Li02] of translating the IMPS library to OMDoc. Our work differs from Li's previous attempt in that it does not try to translate from *only* the internal IMPS data structures. Instead, our implementation *also* reads the corresponding source files to extract additional structure from them. This allows us to compare and corroborate data from one direction of inquiry with data from the other.

Concretely we extend and adapt Li's export mechanism to create a JSON representation of the internal IMPS data structures. Both this and the original source files of the mathematical library are then parsed by our importer extension to MMT to create a structured and typed representation of almost all mathematical objects in the source files, with the only notable exception being proof scripts and macetes (theory-aware tactics).

This representation can then easily be translated into the OMDoc/MMT language, with a formalisation of the foundational logic of IMPS (called LUTINS, see Section 2.1) serving as a formal basis for the translation. The generated output is verified (i.e. type-checked wrt. the **LF** meta-logic [HHP93]) by the MMT system against the implementation of the underlying logic LUTINS to establish a certain, if partial, level of correctness of the translation progress.

This two-layered implementation has the benefit of future-proofing the OMDoc export against potential changes in the format.

The OMDoc/MMT output of the translation not only offers a semantically self-contained archive format, it could also be used in various ways by mathematical knowledge management systems or function as a reference point for other knowledge in a (partially) shared meaning space.

Related Work There have been multiple attempts at translating libraries from one theorem proving system to another in an ad-hoc manner. Examples in-

clude translations from HOL Light to Coq [KW10], from Isabelle/HOL to Isabelle/ZF [KS10] (benefiting from the shared logical framework), from HOL to Isabelle/HOL [OS06] and from Mizar to Isabelle [KPU16].

The translation approach using OMDoc/MMT has been previously used (and shown to be successful) in a number of importers for the MMT system for different mathematical systems, such as PVS [Koh+17], Mizar [Ian+13] and HOL Light [KR14], as part of the OAF (**O**pen **A**rchive of **F**ormalisations) project [OAF].

In all of these, the underlying logical foundation of the system has first been formalised natively in OMDoc/MMT as part of the LATIN library – an OMDoc-based atlas of formal logics, type theories, foundations and various translations between them ([Cod+11; Rab14], available online at [LATIN]). The resulting theory is then used as a meta-theory for importing the corresponding libraries. These imports tend to focus on translating the *statements* of theorems only, and pay less attention to the *proofs*, since proofs are often highly system-specific and difficult to translate without also reproducing all of the machinery of the system in question.

Li previously made an attempt to translate the IMPS math library to OMDoc in [Li02]. This is incomplete in a number of ways: In particular, Li's approach did not handle quasi-constructors (a unique and important feature of IMPS, see Section 3.2) and other important aspects (like theory morphisms with additional assumptions, see Section 3.2), often because they are not represented in a useful manner in the internal data structures. Furthermore, Li was only able to check the *syntactic* validity of the generated XML, which makes the faithfulness of the translation difficult to judge. Finally, there have been a number of substantial representational changes from pure OMDoc to OMDoc/MMT, which renders this translation unusable.

Overview This paper is a refined and condensed version of [Bet18], to which we refer for details and code. In Section 2, we recap all involved systems, including MMT, IMPS and OMDoc. After that, we outline the general idea and some theoretical as well as implementation-related specifics of the translation process in Section 3. Section 4 presents some applications for the OMDoc/MMT library and Section 5 concludes the paper.

2 Preliminaries

2.1 Preliminaries: LUTINS

LUTINS (pronounced as in French, short for "**L**ogic of **U**ndefined **T**erms for **I**nference in a **N**atural **S**tyle") is the underlying logic of the IMPS system.

LUTINS is a variant of Church's simple theory of types [Chu40]. It was developed to allow computerised mathematical reasoning that closely follows mathematical practise as performed by mathematicians "in the wild". And since standard mathematical reasoning often focuses on functions, their properties and

operators on them, LUTINS allows for partial functions, and features a (partial) definite description operator as well as a system of subtypes.

LUTINS is a classical logic in the sense that it allows non-constructive reasoning, but non-classical in the sense that terms in LUTINS can be non-denoting. It also supports λ-notation for functions, an infinite hierarchy of function types for higher-order functions, and full quantification (existential and universal) over all function types.

Languages, Sorts, and Expressions The notion of languages is central to LUTINS. They contain two classes of objects: sorts and expressions. Sorts denote (non-empty) domains of mathematical objects and expressions denote members of these domains. Expressions can be used to directly reference mathematical objects and to make statements about them using a LUTINS language given by a set of sort declarations and (sorted) constant declarations (see [FGT98]).

We differentiate between *atomic sorts* (e.g. `ind`, `zz`, ...) and *compound sorts*, the latter denoting the domain of n-ary functions for an arbitrary n (e.g. $[\texttt{zz}, \texttt{ind}]$ for $n = 2$). Sorts may overlap, but they cannot be empty. Every language includes the base type $\star$ (sometimes also defined as $*$ and always denoted as such in the implementation), denoting the set $\{\texttt{T}, \texttt{F}\}$ of standard truth values.

Sorts are also divided into two *kinds*, $\star$ (read: *star* or *prop*) and ι (read: *ind*). A given sort α is of kind $\star$ if either $\alpha = \star$ or α is a compound sort *into* $\star$ (i.e. a compound sort of the form $[\alpha_1, \ldots, \alpha_n, \star]$, sometimes also called a *predicate*). In *all* other cases α is of kind ι. This includes all atomic sorts except $\star$ itself.

LUTINS allows for sorts to be defined as *subsorts* of other sorts in multiple ways. For instance, the natural numbers $\mathbb{N}$ form a subsort of the real numbers $\mathbb{R}$ and the continuous (real) functions a subsort of the functions from $\mathbb{R}$ to $\mathbb{R}$.

Each atomic sort is assigned a unique *enclosing sort* by the language that defines it. This gives rise to a particular partial order on its sorts, which we will call $\preceq$ (also sometimes called "the subsort relation") that is intended to denote set inclusion. A sort that is maximal in relation to $\preceq$ is called a *type*. The type of a given sort α has the notation $\tau(\alpha)$.

Subsorting also applies to compound sorts. In particular, if $\sigma_0 \preceq \tau_0$ and $\sigma_1 \preceq \tau_1$, then $[\sigma_0, \sigma_1] \preceq [\tau_0, \tau_1]$. This makes subsorting in LUTINS *co*variant in its arguments, not *contra*variant as is common in settings without partial functions. The compund sort $[\sigma_0, \sigma_1]$ contains exactly those partial functions that are never defined outside of σ_0 and never return values outside σ_1. For example, you could pass any real number to a function expecting a natural number (given that $\mathbb{N}$ and $\mathbb{R}$ have the same type). If the number is indeed not a natural number, the expression will be undefined (see below).

All of this is helpful for mechanised deduction because the subsorting relation can give important information about the value of an expression, should it be defined. Furthermore, many theorems have constraints that can easily be expressed in terms of a subtype and the prover can be programmed to handle these with special algorithms.

Partial Functions, Undefined and Non-Denoting Values The stated goal of IMPS (and therefore LUTINS) is to allow for reasoning that is very close to mathematical practice. This means that there needs to be a way to deal with partial functions and undefined values since these make frequent appearances in chalk-and-whiteboard mathematics. For example, all of the terms $\frac{5}{0}, \sqrt{-3}, \ln(-4)$ are undefined in the standard theory of arithmetic over the real numbers.

Note that there is a subtle difference between a term that is "undefined" and one that is "non-denoting". According to Farmer, a term is undefined if it is not assigned a "natural" meaning and non-denoting if it is not to be assigned any meaning at all. Often, an undefined term is also non-denoting, but it *can* still have a denotation. For example, the term $\frac{5}{0}$ does not have a "natural meaning" in standard real arithmetic, but is sometimes assigned a value in practice anyway. In particular, IMPS follows the approach of *partial valuation for terms but total valuation for formulas* (see [Far90] for more details). This means a term of type $\star$ *always* has a denotation (if one of its constituents is undefined, that denotation is F).

Definite Description One of the more prominent features of LUTINS is the possibility of reasoning with definite description via the ι (or *iota*) constructor. Given a variable v of sort α of *kind* ι (not to be confused with the constructor itself) and an unary predicate φ over α, the expression $\iota\, v : \alpha\,.\,\varphi(v)$ denotes the *unique* v, such that $\varphi(v)$, if there exists such a v. If there is no or more than one v that fulfils the predicate, the ι-expression is undefined.

For example, the expression $\iota\ x : \mathbb{R}.(0 \leq x) \wedge (x \cdot x = 2)$ denotes $\sqrt{2} \in \mathbb{R}$, while the expression $\iota\ x : \mathbb{R}.x \cdot x = 2$ is undefined.

Definite description can be very useful for dealing with functions, especially partial functions, which is why it is featured so prominently in IMPS.

2.2 Preliminaries: IMPS

IMPS (short for "Interactive Mathematical Proof System") is an interactive theorem prover developed by William Farmer, Joshua Guttmann and Javier Thayer from 1990 to 1993 [TMI]. It was one of the influential systems in the era of automated reasoning.

One of the goals in developing IMPS was to create a mathematical system that gave computational support to mathematical techniques common among actual mathematicians.

The development of the IMPS system has been heavily influenced (see [FGT98]) by three insights into real-life mathematics:

- Mathematics emphasises the *axiomatic method*. The characteristics of mathematical structures are captured in axioms. Theorems are then derived from these axioms for *all* structures that satisfy the axioms.
  Often, what is needed for a proof is a clever change of perspective to see that one structure is indeed an instance of another theory, bringing additional theorems to bear.

– Many branches of mathematics emphasise *functions*, including partial functions. Moreover, the classes of objects studied may be nested, as are the integers and the real numbers; or overlapping, as are the bounded functions and the continuous functions.
– Mathematical proofs usually employ a mixture of both *formal inference* and *computation*.

Special attention is directed at the interplay of *computation* and *proof*. Farmer, Guttman and Thayer emphasise that, for example, a mathematician might devote considerable effort into proving lemmas that justify computational procedures[1] but are ultimately uninterested in the part of the derivation that is the "implementation" of these procedures.

Therefore, IMPS also allows for inferences based on sound computation and not merely formal inference. These are treated as atomic inferences, although a full formalisation in – for example – a Gentzen-style system might require hundreds or thousands of inference steps.

Little Theories When following the axiomatic method to do mathematics – that is, logically reasoning from a given set of sentences in a formal language – there are two prominent approaches to chose from, which we will refer to as the "little theories" and "big theories" approach.

In the "big theories" version of the axiomatic method, all reasoning is carried out in one highly expressive axiomatic theory. The set of axioms selected is powerful enough, such that any model of them will contain all the mathematical objects that are of interest to us, and deduction from these powerful axioms will be enough to prove the relevant theorems in the theory. Popular examples for a "big" axiomatic theory would be ZFC or the Calculus of Inductive Constructions.

Contrasted with that, the "little theories" approach uses a number of different theories with smaller, less powerful sets of axioms, to develop mathematics in. For example, one theorem could be true for all semi-rings, while another is only true in the theories of *commutative* rings. Theorems are proved by logical derivation from the axioms of whatever theory supplies the necessary structure for the proof.

Both IMPS and MMT subscribe to the "little theories" approach to formal mathematics, a design choice that was informed by the fact that the little theories approach lends itself well to the mechanism of theory interpretations [FGT92].

Theories are the basic unit of representing mathematical knowledge in IMPS. In fact, Farmer (in [FGT98]) calls IMPS "a system for developing, exploring, and relating theories".

Theory Morphisms A *theory morphism* (sometimes also called a *theory interpretation*) is a translation between two theories that maps expression from the one theory to expressions in the other, with the additional property that theorems are always mapped to theorems ([Far93] and [FGT98]).

[1] [FGT98] gives the example of the algorithm for differentiating polynomials for this.

This is an integral part of the "little theories" approach as theory morphisms are the tool to use to make results of one theory available in the other.

It is also close to mathematical practice, since seeing one structure as an instance of another (and therefore bringing all theorems of the other structure into play) is often the critical insight in non-trivial mathematical proofs.

2.3 Preliminaries: OMDoc/MMT

OMDoc (short for **O**pen **M**athematical **Doc**uments) is a semantics-oriented markup format for STEM-related documents extending OpenMath developed by the KWARC work group (see [Koh06]). OMDoc/MMT [RK13] re-conceptualises the formal/modular fragment of OMDoc and greatly enhances its expressive power. OMDoc/MMT retains OMDoc's three distinct levels for expressions of mathematical knowledge: **Object Level** Expressions (e.g. terms and formulae) expressed in OpenMath, **Declaration Level** Constants (functions, types, judgements) with an optional (object-level) type and/or definition and **Module Level** Theories and Views; sets of declarations that inhabit a common name-space and context.

Theories in OMDoc/MMT are structurally similar to theories in `IMPS` and can include other theories. Hence MMT-theories allow for library development in concordance to the little theories paradigm. Views in MMT behave (for all purposes relevant in this paper) analogously to theory morphisms in `IMPS`.

The MMT System The OMDoc/MMT language is implemented in the MMT system [Rab18], which provides an API to handle OMDoc/MMT content and services such as type checking, rewriting of expressions and computation, as well as notation-based presentation of OMDoc/MMT content and a general infrastructure for inspecting and browsing libraries.

Since OMDoc/MMT avoids committing to a specific semantics or logical foundation, foundation-dependent services and features (e.g. type checking, presentation) are implemented using (foundation-independent) generic algorithms extensible by foundation-dependent calculus rules via plug-ins (e.g. for handling content imported from external systems such as `IMPS`).

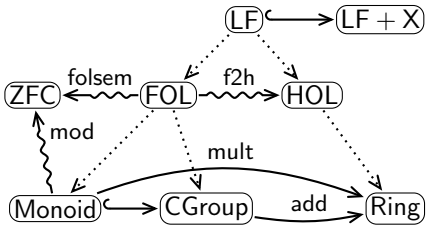

Fig. 1. Meta-Levels in OMDoc/MMT

Theory Graphs Theories and theory morphisms naturally lead to *theory graphs*, with theories as vertices and morphisms as edges. In fact, OMDoc/MMT-theories and morphisms form a category, which is exploited by the MMT-system to induce and translate knowledge in/between theories analogously to `IMPS`).

The possible arrows in OMDoc/MMT are **inclusions**, which import all declarations from the domain to

the co-domain, **views**, which are judgement-preserving maps from the declarations in the domain to expressions over the co-domain, **structures**, which are omitted for this paper, and the **meta-theory**-relation, which behaves like an include for most purposes (The meta-theory-relation connects theories that live on different meta-levels; e.g. domain knowledge to its logical foundation and conversely the logical foundation to the logical framework it is formalised in.).

An example graph is given in Figure 1. Dotted lines represent the meta-theory-relation, hooked arrows are includes, squiggly arrows represent views, and the normal (labelled) arrows represent structures. The MMT system also provides a theory graph viewer (see [RKM17]), an example for which is given in Figure 9. For our purposes, we fix as a foundation the logical framework **LF** (see [HHP93]), since it is particularly well supported by the MMT system.

3 Implementation

3.1 The LUTINS Theory in LF

To formalise LUTINS in MMT, we use the logical framework **LF**, which provides a dependently typed lambda calculus with *i*) two universes `type` and `kind` with `type:kind` and *ii*) dependent function types $\prod_{x:A} T(x)$ (in **LF**-syntax: `{x:A}T(x)`). If T does not contain the variable x, this is the same as the function type $A \to T$. Dependent function types are inhabited by lambda expressions $\lambda x : A.t(x)$ (in **LF**-syntax: `[x:A]t(x)`). The usual rules in a lambda calculus (extensionality, beta-reduction, ...) hold.

To represent LUTINS, we created a **LF** meta-theory[2] that, for every concept in the logic itself (like quantifiers, logical constructors, the primitive sorts, ...), has a corresponding constant (44 of them). Furthermore, we declare:

1. a new **LF**-type `tp:type`, which serves as the universe of maximal IMPS-sorts,
2. a function `sort : tp` $\to$ `type`, and
3. a function `exp : {A : tp} sort A` $\to$ `type`.

Given some maximal IMPS-sort `A`, the **LF**-type `sort A` then serves as the type of all subsorts of that IMPS-type, and given a sort `a : sort A`, the type `exp A a` corresponds to the **LF**-type of all IMPS-expressions of sort `a`.

We use the principles of *higher-order abstract syntax* to specify *binders* in IMPS. For example, consider an IMPS expression $\lambda x : A.t$, where the λ-constructor binds a new variable $x : A$. We formalise this behaviour by declaring the IMPS lambda to be an **LF** function `lambda`, that takes an **LF** lambda expression as argument which binds the variable x. As a result we get the **LF** expression `lambda ([x:A] t)` being the application of the function `lambda` to the **LF** function `[x:A]t`, effectively "embedding" an **LF** function on IMPS expressions as an IMPS function. Application in IMPS, quantifiers and other binders are treated analogously.

[2] This formalisation is part of the LATIN foundations see `https://gl.mathhub.info/MMT/LATIN/blob/master/source/foundations/imps/lutins.mmt`

For propositional judgements (i.e. axioms and theorems) in IMPS, we use the *judgements-as-types* paradigm by introducing an operator `thm : exp bool → type`, assigning to each proposition a type which we can think of as the "type of proofs" for that proposition. Correspondingly, we consider a proposition A to be "true" if the type `thm` A is inhabited. Axioms correspond to undefined constants of type `thm A`, whereas theorems correspond to defined constants of that type, their definition being a proof (although proofs are omitted in this paper).

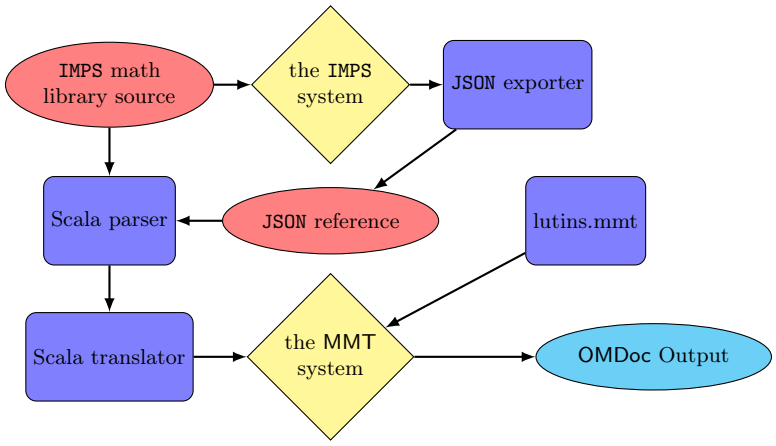

Fig. 2. Overview: Red: Source Files, Blue: Our contributions, Yellow: Independent systems Cyan: Resulting OMDoc

3.2 Translation

We now present the actual transformation process. It starts with IMPS library files[3] and uses several software systems over a number of different steps, which are outlined below. Figure 2 gives a high-level schematic view of all involved systems and processes. The individual steps of the translation process are as follows:

- **Generate JSON from IMPS data structures** For this, we modified Li's exporter to export (the relevant aspects of) the internal data structures in IMPS directly to JSON, which is easy to read (for both human and machine) and gives us direct access to the data in the internal data structures, instead of an outdated OMDoc *translation* of those structures.
- **Import and combine JSON and IMPS sources** Parsing from both IMPS library source files and JSON generated from internal data structures, gives the possibility of including more data in the translation, even data that is not represented on a symbolic level within IMPS.

[3] Which – like the original IMPS system – are written in the T language – a dialect of Scheme – and are hence often referred to simply as "T-files" in the following sections.

– **Translate combined structures to MMT/OMDoc** The last step uses the **LF**-implementation of LUTINS. In this form, they can also be type-checked by MMT to verify their correctness. The final OMDoc output is also generated by the MMT system, which always produces OMDoc in the *current* standard of the format.

```
(def-atomic-sort nn              ;;; Name
  "lambda(x:zz, 0<=x)"           ;;; Defining Expression
  (theory h-o-real-arithmetic)   ;;; Home Theory
  (witness "0"))                 ;;; Witness to show the sort non-empty
```

Fig. 3. `IMPS` Source Code: Def-Form defining the atomic sort `nn` via predicate

Def-Forms `IMPS` source files contain information in so-called "def-forms" (short for *"definition forms"*). Each def-form is essentially the specification of one `IMPS` object, from constants, theories, languages to translations. Figure 3 shows an example.

Group	Amount	`foundation`	`imps-math-library`
foundational	17	414 (100%)	1918 (93.8%)
advanced	10	0 (0%)	119 (6.2%)
unused	5	0 (0%)	0 (0%)

Fig. 4. Survey results for usage of each def-form

To avoid unnecessary work in implementation, we did a survey of the `imps-math-library` to determine which def-forms were used how often. In Figure 4 we consider a def-form *"unused"*, if it does not appear in the library, even if it *is* supported by `IMPS`. We classify a def-form as *"foundational"* if it appears in the `foundation` sub-library. All other def-forms (called *"advanced"* in this context) were initially given low priority.

S-Expressions There are different ways of representation in which `IMPS` displays mathematical objects to the user. One of the most important features of the `JSON` export mechanism is the export of mathematical expressions in *s-expression syntax* (as popularised by LISP) instead of *string syntax*.

For example, consider the axiom `commutative-law-for-addition` of the theory `h-o-real-arithmetic`. In string presentation, it is printed like this:

```
forall(y,x:rr,x+y=y+x)
```

In s-expression syntax, however, this axiom is printed as follows:

```
(forall ((rr y x)) (= (apply-operator + x y)
                      (apply-operator + y x)))
```

While the string representation might be more familiar to the human eye, s-expressions are considerably easier to parse mechanically and make dealing with binding strength and operator precedence unnecessary. They also simplify parsing function applications and quasi-constructors.

Quasi-Constructors In addition to the LUTINS core logical constructors it is also possible for a user of IMPS to define additional constructor-like forms called "quasi-constructors". These are implemented as "macros" or "abbreviations".

For example, in IMPS, there exists the notion of quasi-equality: two expressions are quasi-equal if and only if they are either both undefined or are both defined with the same value. In mathematical notation, this would be captured by the following biconditional:

$$E_1 \simeq E_2 \equiv (E_1{\downarrow} \vee E_2{\downarrow}) \supset E_1 = E_2$$

More precisely, a quasi-constructor consists of three elements: a *name* (something like `quasi-equals`), a list of variables (E_1 and E_2) and a schema (the right hand side above, see Figure 5 for an example from the library).

In addition to the user-defined quasi-constructors, the IMPS system also has a small number of so-called "system quasi-constructors" that are hard-wired into the deductive machinery. Quasi-equality is one of them. Quasi-constructors are polymorphic in their schema variables, even if this polymorphism is not made explicit in the notation.

The translation of quasi-constructors turned out to be quite challenging. As Li states in [Li02], the corresponding lambda expressions for quasi-constructors are not represented as symbols in IMPS and can therefore not be translated into JSON directly, like other expressions.

```
(def-quasi-constructor I-IN
    "lambda(x:uu,a:sets[uu], #(a(x)))"
     (language indicators)
     (fixed-theories the-kernel-theory))
```

Fig. 5. The quasi-constructor `i-in`, as declared in IMPS

However, user-defined quasi-constructors are used *extensively* throughout the source. A survey of the T source files and the JSON output of just the `foundation` section identified 58 quasi-constructors used hundreds of times within the library-section `imps-math-library`. Thus any effort to translate this library would be incomplete without a rigorous treatment of quasi-constructors.

Instead of manually adding each individual quasi-constructor to the theory that defines it, or automatically resolving them immediately when parsed (which would need a lot of typing information not easily available at that stage of the translation) we decided to formulate one global **LF**-theory (called *QuasiLutins*) for them. There, we implemented all quasi-constructors as an instance of the same data type that also represents ordinary constructors (as seen in Figure 6).

```
inQC : { A, α : sort A } exp α ⟶ exp (sets[α]) ⟶ exp bool
     = [U,u,x,a] (a @ x) ↓
```

Fig. 6. The same quasi-constructor, implemented in **LF**

This turned out to be the most effective and most faithful approach to the original sources, since the separation of theories makes clear what is part of

the original LUTINS and what is not. It is also possible to stick to genuine polymorphism this way, without having to re-derive too much typing information during translation.

Theory Morphisms IMPS translations (which are all *interpretations* in the IMPS library, i.e. all the obligations of the translation are theorems in the target theory) are translated as MMT *views*.

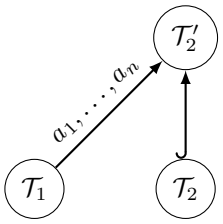

Fig. 7. Theory morphisms with axioms

Some theory morphisms in IMPS (see Figure 8 for an example showing the translation of groups to subgroups) have a collection of assumptions that need to be fulfilled (i.e. need to be theorems in the target theory for the morphism to be applicable). These assumptions can be used to state that certain conditions must be met (e.g. in the example from above, the target set (indicator function) must not be empty).

Views in MMT, however, are not designed to have assumptions. To circumvent this obstacle, we create a copy of the target theory $\mathcal{T}_2$, called $\mathcal{T}_2'$ that includes $\mathcal{T}_2$, but also has all the assumptions associated with the theory morphism as additional axioms (see Figure 7).

```
(def-translation GROUPS->SUBGROUP
  (source groups)
  (target groups)
  (assumptions
   "with(a:sets[gg], nonempty_indic_q{a})"
   "with(a:sets[gg], forall(g,h:gg, (g in a) and (h in a)
                                        implies (g mul h) in a))"
   "with(a:sets[gg], forall(g:gg, (g in a) implies (inv(g) in a)))")
  (fixed-theories h-o-real-arithmetic)
  (sort-pairs
   (gg (indic "with(a:sets[gg], a)")))
  (constant-pairs
   (mul "with(a:sets[gg], lambda(x,y:gg, if((x in a) and (y in a),
                                           x mul y, ?gg)))")
   (inv "with(a:sets[gg], lambda(x:gg, if(x in a, inv(x), ?gg)))"))
  force-under-quick-load
  (theory-interpretation-check using-simplification))
```

Fig. 8. IMPS Source Code (from "subgroups.t"): Subgroup Translation

4 Applications

Continued Theory Library Development Translating the IMPS theory library into OMDoc/MMT format allows us to use the theories contained therein in future projects and enables the continuing development of other theories that

build upon them (without depending directly on the IMPS system itself). They are now also available to other tools and automated methods (e.g. as data for machine-learning approaches to auto-formalisation).

Alignments Using flexible alignments (see [Mül+17]) between different libraries (such as those of the PVS, HOL Light, and Mizar projects, for which there exist similar translation efforts), we can guide library developers to corresponding parts of other formalisations, give decent approximate translations of content across libraries, or help users more familiar with IMPS towards content of other systems by re-using notations that would otherwise be system specific.

OMDoc/MMT Services With the OMDoc/MMT translation of the IMPS theory library, IMPS also gains access to library management facilities implemented at the OMDoc/MMT level. There are two ways to exploit this: publishing the translated IMPS libraries on a dedicated server, like the MathHub system, or running the OMDoc/MMT stack locally.

Browsing and Interaction The transformed IMPS content can be browsed interactively in the document-oriented MathHub presentation pages (theories as active documents) and in the MMT web browser. Both allow interaction with the IMPS content via a generic Javascript-based interface.

Graph Viewer The MMT system includes a theory graph viewer [RKM17] that allows interactive, web-based exploration of the OMDoc/MMT theory graphs. It builds on the `vis.js` JavaScript visualisation library, which uses the HTML5 canvas to layout and interact with graphs client-side in the browser.

The IMPS theory library relies substantially on theories as a structuring mechanism (as a consequence of taking the *little theories* approach), which makes a graph viewer particularly attractive. Figure 9 shows the full graph of the `foundation` library section, generated from only the OMDoc translated from IMPS[4].

5 Conclusion

We have developed a representation of the IMPS logic LUTINS and an automated translation of the IMPS mathematical theory library in the OMDoc/MMT format. This saves the IMPS library from be coming inaccessible and allows continued development and cross-fertilisation.

This information architecture is essential for system interoperability. In our case we have shown that we can use the language-independent MMT tool chain for IMPS. In particular, with the library browser and the theory graph viewer, we have instantiated two generic periphery systems for IMPS.

[4] Note that the theories shown here are all part of the library; they are not duplicates created by the process from Figure 7.

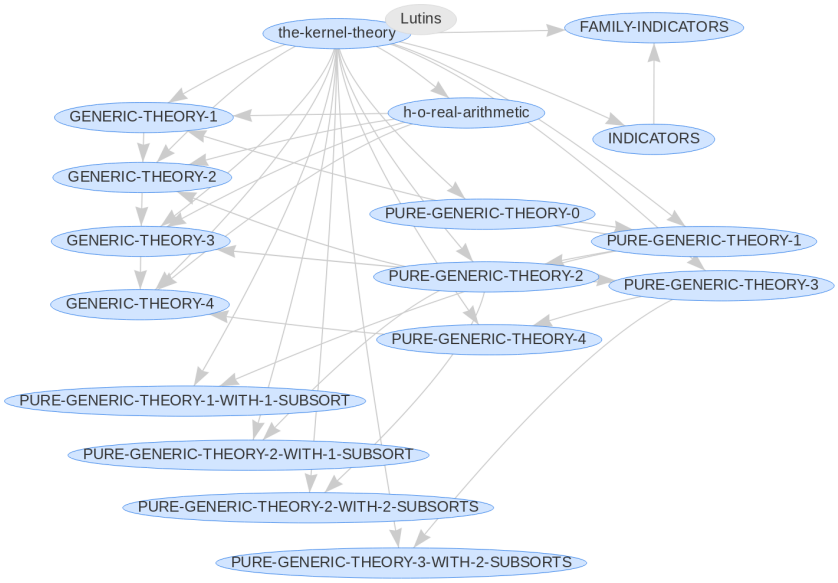

Fig. 9. Theory Graph of the `foundation` section

Future Work Our results can also easily be extended to use LFX (**LF** + X, an extension to the **LF** framework, see [LFX]) to give shallower (i.e. more structure-preserving) encodings of `IMPS` features without having to sacrifice the advantages of logical frameworks via the use of *structural features* (see [Ian17]).

Finally, in future efforts, we would like to extend the current export to also include proofs and macetes of the `IMPS` system as non-opaque data. For this to be possible, we would have to represent the `IMPS` proof calculus in **LF**, and develop a **LF** representation for proof commands. In our experience, both tactic-level proof scripts as well as full proofs are even harder to make interoperable than the statement level of libraries and may thus be less useful.

Software Sources All software that is mentioned in this paper is available online: *i*) **imps2json**: `https://gl.mathhub.info/IMPS/theories` *ii*) **MMT extension**: `https://github.com/UniFormal/MMT/tree/imps` *iii*) **MMT archive**: `https://gl.mathhub.info/IMPS/imps`

Acknowledgments The authors gratefully acknowledge financial support from DFG-funded project OAF: An Open Archive for Formalizations (KO 2428/13-1) and fruitful discussions and clarifications from Bill Farmer, Dennis Müller, and Florian Rabe.

References

[Bet18] Jonas Betzendahl. "Translating the IMPS Theory Library to MMT / OMDoc". Master's Thesis. Informatik, Universität Bielefeld, Apr. 2018. URL:

https://gl.kwarc.info/supervision/MSc-archive/blob/master/2018/jbetzendahl/thesis_imps2omdoc.pdf.

[Chu40] Alonzo Church. "A Formulation of the Simple Theory of Types". In: *Journal of Symbolic Logic* 5 (1940), pp. 56–68.

[Cod+11] Mihai Codescu, Fulya Horozal, Michael Kohlhase, Till Mossakowski, and Florian Rabe. "Project Abstract: Logic Atlas and Integrator (LATIN)". In: *Intelligent Computer Mathematics*. Ed. by James Davenport, William Farmer, Florian Rabe, and Josef Urban. LNAI 6824. Springer Verlag, 2011, pp. 289–291. URL: https://kwarc.info/people/frabe/Research/CHKMR_latinabs_11.pdf.

[Far90] William M. Farmer. "A partial-function version of Church's simple theory of types". In: *Journal of Symbolic Logic* 55 (1990), pp. 1269–1291.

[Far93] William M. Farmer. "Theory Interpretation in Simple Type Theory". In: *HOA'93, an International Workshop on Higher-order Algebra, Logic and Term Rewriting*. LNCS 816. Amsterdam, The Netherlands: Springer Verlag, 1993.

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

[FGT98] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. *The IMPS 2.0 User's Manual*. 1st ed. The MITRE Corporation. Bedford, MA 01730 USA, Jan. 1998.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. "A framework for defining logics". In: *Journal of the Association for Computing Machinery* 40.1 (1993), pp. 143–184.

[Ian+13] Mihnea Iancu, Michael Kohlhase, Florian Rabe, and Josef Urban. "The Mizar Mathematical Library in OMDoc: Translation and Applications". In: *Journal of Automated Reasoning* 50.2 (2013), pp. 191–202. DOI: 10.1007/s10817-012-9271-4.

[Ian17] Mihnea Iancu. "Towards Flexiformal Mathematics". PhD thesis. Bremen, Germany: Jacobs University, 2017. URL: https://opus.jacobs-university.de/frontdoor/index/index/docId/721.

[Koh+17] Michael Kohlhase, Dennis Müller, Sam Owre, and Florian Rabe. "Making PVS Accessible to Generic Services by Interpretation in a Universal Format". In: *Interactive Theorem Proving*. Ed. by Mauricio Ayala-Rincón and César A. Muñoz. Vol. 10499. LNCS. Springer, 2017. URL: http://kwarc.info/kohlhase/submit/itp17-pvs.pdf.

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

[KPU16] Cezary Kaliszyk, Karol Pąk, and Josef Urban. "Towards a Mizar Environment for Isabelle: Foundations and Language". In: *Proc. 5th Conference on Certified Programs and Proofs (CPP 2016)*. Ed. by Jeremy Avigad and Adam Chlipala. ACM, 2016, pp. 58–65. DOI: 10.1145/2854065.2854070.

[KR14] Cezary Kaliszyk and Florian Rabe. "Towards Knowledge Management for HOL Light". In: *Intelligent Computer Mathematics 2014*. Ed. by Stephan Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban. LNCS 8543. Springer, 2014, pp. 357–372. URL: http://kwarc.info/frabe/Research/KR_hollight_14.pdf.

[KS10] Alexander Krauss and Andreas Schropp. "A Mechanized Translation from Higher-Order Logic to Set Theory". In: *Interactive Theorem Proving: First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*. Ed. by Matt Kaufmann and Lawrence C. Paulson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 323–338. DOI: 10.1007/978-3-642-14052-5_23.

[KW10] Chantal Keller and Benjamin Werner. "Importing HOL Light into Coq". In: *ITP*. Vol. 6172. Lecture Notes in Computer Science. Springer, 2010, pp. 307–322.

[LATIN] *The LATIN Logic Atlas*. URL: https://gl.mathhub.info/MMT/LATIN (visited on 06/02/2017).

[LFX] *MathHub MMT/LFX Git Repository*. URL: http://gl.mathhub.info/MMT/LFX (visited on 05/15/2015).

[Li02] Yan Li. "IMPS to OMDoc Translation". Bachelor's Thesis. McMaster University, Aug. 2002.

[Mül+17] Dennis Müller, Thibault Gauthier, Cezary Kaliszyk, Michael Kohlhase, and Florian Rabe. "Classification of Alignments between Concepts of Formal Mathematical Systems". In: *Intelligent Computer Mathematics (CICM) 2017*. Ed. by Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke. LNAI 10383. Springer, 2017. DOI: 10.1007/978-3-319-62075-6.

[OAF] *The OAF Project & System*. URL: http://oaf.mathhub.info (visited on 04/23/2015).

[OS06] Steven Obua and Sebastian Skalberg. "Importing HOL into Isabelle/HOL". In: *Automated Reasoning: Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006. Proceedings*. Ed. by Ulrich Furbach and Natarajan Shankar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 298–302. DOI: 10.1007/11814771_27.

[Rab14] Florian Rabe. "How to Identify, Translate, and Combine Logics?" In: *Journal of Logic and Computation* (2014). DOI: 10.1093/logcom/exu079.

[Rab18] Florian Rabe. "MMT: A Foundation-Independent Logical Framework". Online Documentation. 2018. URL: https://kwarc.info/people/frabe/Research/rabe_mmtsys_18.pdf.

[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.

[RKM17] Marcel Rupprecht, Michael Kohlhase, and Dennis Müller. "A Flexible, Interactive Theory-Graph Viewer". In: *MathUI 2017: The 12th Workshop on Mathematical User Interfaces*. Ed. by Andrea Kohlhase and Marco Pollanen. 2017. URL: http://kwarc.info/kohlhase/papers/mathui17-tgview.pdf.

[TMI] *Theorem Prover Museum – IMPS*. URL: https://github.com/theoremprover-museum/imps (visited on 04/28/2018).