Alignment-based Translations Across Formal Systems Using Interface Theories

Dennis Müller
Computer Science, FAU Erlangen-Nürnberg

Colin Rothgang
Mathematics, Jacobs University Bremen

Yufei Liu
Mathematics, Jacobs University Bremen

Florian Rabe
Computer Science, Jacobs University Bremen

Translating expressions between different logics and theorem provers is notoriously and often prohibitively difficult, due to the large differences between the logical foundations, the implementations of the systems, and the structure of the respective libraries. Practical solutions for exchanging theorems across theorem provers have remained both weak and brittle. Consequently, libraries are not easily reusable across systems, and substantial effort must be spent on reformalizing and proving basic results in each system. Notably, this problem exists already if we only try to exchange theorem *statements* and forgo exchanging *proofs*.

In previous work we introduced alignments as a lightweight standard for relating concepts across libraries and conjectured that it would provide a good base for translating expressions. In this paper, we demonstrate the feasibility of this approach. We write interface theories in a foundationally uncommitted framework that abstract from logical foundation, implementation, and library structure. Then we use alignments to record how the concepts in the interface theories are realized in several major proof assistant libraries, and we use that information to translate expressions across libraries. Concretely, we present exemplary interface theories for several areas of mathematics and — in total — several hundred alignments that were found manually.

1 Introduction and Related Work

Motivation There is a vast plurality of formal systems and corresponding libraries for formalized knowledge. However, almost all of these are non-interoperable because they are based on differing, mutually incompatible logical foundations (e.g. set theory, higher-order logic, variants of type theory etc.), implementations, and library structures, and much work is spent developing the respective basic libraries in each system. Because a library in one such system is not reusable in another system, developers are forced to spend a large amount of time and energy developing parallel formalizations in multiple systems.

Ideally, given two or more libraries, one would want to integrate them with each other so that knowledge formalized in one of them can be reused in, translated to or identified with contents of the others. In particular, translation across formal systems offers up many potential applications such as

- accessing and using contents from other libraries during formalization,
- aiding premise selection by importing the premises used in proofs from other systems,
- presenting contents of an unfamiliar library using notations of a familiar one, or
- freely combining user interfaces with libraries from different systems.

Integrating Libraries There are two ways to take on library integration. Firstly, we can try to translate the contents of one formal system directly into another system. This requires intimate knowledge of both systems and their respective foundations used.

To appear in EPTCS.

Secondly, we can use a *logical framework* that provides a uniform intermediate data structure, in which we can specify the respective foundations and their libraries. This approach has been used by the authors' research group in [IKR11] for Mizar, in [KR14] for HOL Light and in [Koh+17a] for PVS, using the MMT framework [RK13]. A similar approach is currently underway using Dedukti [BCH12] as the framework system.

Neither solution has proved particularly successful. On the one hand, direct translations are expensive and must be developed for each pair of systems. On the other hand, logical frameworks are a step in the right direction because they allow for a star-shaped integration architecture. But they do not magically solve the library integration problem: logic and library translations remain extremely difficult. A detailed analysis is given in [KR16].

In recent work we have come to understand this problem more clearly and suggested a systematic solution. Firstly, in [KRSC11] and [KR16], we developed the idea of **interface theories**. Using an analogy to software engineering, we can think of interface theories as specifications and of theorem prover libraries as implementations of formal knowledge. Libraries of interface theories must critically differ from typical theorem prover libraries: they must follow the axiomatic method (as opposed to the method of definitional extensions), be written with minimal foundational commitment, and not contain definitions or proofs.

Secondly, in [Kal+16], we introduced **alignments** as a primitive concept for lightweight translations between libraries. In the simplest case, an alignment is a pair of symbol identifiers from two different libraries such that both symbols are "morally the same" mathematical concept. In particular, two aligned symbols may use entirely different (possibly not even logically equivalent) definitions. Specifically, alignments between an interface theory and a theorem prover provide the information how a library implements the concepts of the interface theory.

Thirdly, in the OpenDreamKit project [CICM1616; ODKb] we pursue the same approach in the context of computer algebra systems. In this context, we have already developed some interface theories for basic logical operations such as equality, with approximately 300 alignments to theorem prover libraries.

In this paper we follow up on this suggestion by designing several interface theories and finding their alignments to several major formal libraries. Concretely, we have built theories for numbers, sets, lists, topology, combinatorics and analysis as representative examples of important domains. Moreover, we have found alignments of the involved symbols with the libraries of HOL Light, PVS, and Mizar .

Consider for example the PVS symbol `member` and the HOL Light symbol `IN`. Both represent the mathematical concept of element-hood of (typed) sets, but they live in different libraries based on different foundations employed by different systems.

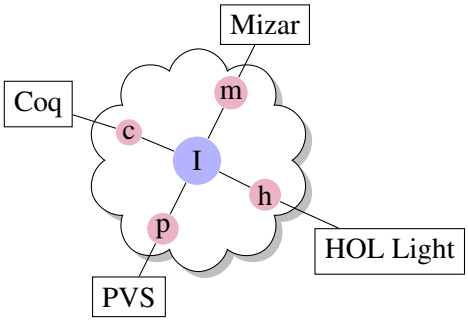
We can see the two symbols being aligned via a sequence of consecutive steps of abstraction:

1. We import the libraries of formal systems to a generic, foundationally neutral formal language (namely OMDoc/MMT) with the help of a formalization of the system's primitives.
2. We align the system specific symbols (which we collectively refer to as the *system dialect*) with generic representations of the same concept. E.g. we align a symbol for HOL Light's function-application with a generic symbol for function application.
3. Ultimately, we (possibly after additional steps) end up with the same purely mathematical concept expressed in a system- and foundationally neutral language.

For the example with sets, we capture this with

- a symbol for elementhood in the interface theory for sets,
- two alignments of this new symbol with the symbols `member` and `IN`, respectively.

This yields a star-shaped network as in the diagram on the right with various formal libraries on the outside, which are connected by alignments via representations of their system dialects (lower-case letters) to the interface theory in the center.



Finding Alignments Even though we know that numerous alignments exist between libraries, not many alignments are known concretely or have been represented explicitly. Therefore, a major initial investment is necessary to obtain a large library of interface theories and alignments. There are three groups of alignment-finding approaches.

Human-based approaches examine libraries and manually identify alignments. This approach has been pursued ad hoc in various contexts. For example, the library translation of [OS06] included some alignments for HOL Light and Isabelle/HOL, which were later expanded by Kaliszyk. The Why3 and FoCaLiZe systems include alignments to various theorem provers that they use as backends.

The remaining two classes use *artificial intelligence* methods.

Logical approaches align two concepts if they satisfy the same theorems. This cannot directly appeal to logical equivalence because that would require a translation between the corpora. Instead, they compare the theorems that are explicitly stated in the corpora. The theorems should be normalized first to eliminate differences that do not affect logical equivalence. The quality of the results depends on how completely those theorems characterize the respective concepts. In well-curated libraries, this can be very high [MK15].

Machine learning–based approaches are inherently based on statistical patterns and hence naturally inexact. The main research in this direction is carried out by Kaliszyk and others [GK14].

While finding alignments automatically is promising for *perfect alignments*, where two symbols only differ in name but are otherwise used identically, it is a different matter for *imperfect* alignments. For example, consider binary division which yields undefined or 0 when the divisor is zero, versus a strict division that requires an additional proof-argument that the divisor is nonzero. Here automation becomes much more difficult, because the imperfections often violate the conditions that an automatic approach uses to spot alignments.

We apply the human-based approach in this paper and demonstrate that it is feasible. We present the largest set ever of systematically collected, human-verified alignments from multiple major proof assistants. In addition to its inherent value, it will also greatly benefit the artificial intelligence approaches: firstly, it provides a dataset that can be used for evaluation and training; secondly, we conjecture that the quality of artificial intelligence approaches will be massively improved if they are applied on top of a large set of guaranteed-perfect alignments. This is based on two observations:

- The more alignments we know, the easier it is to find new ones. Consider a typical formalization in system S_1 that introduces a new concept c relative to some known ones. If perfect alignments to system S_2 for the known concepts have already been established, it becomes relatively easy to find the formalization of c in S_2, and add an alignment for it.
- It is very difficult to get alignment-finding off-the-ground. Because the foundations of S_1 and S_2 are often very different, almost nothing looks particularly similar in the absence of any alignments. Deeper alignments will become more apparent only when alignments for fundamental concepts such as booleans, sets and numbers are established.

Leveraging Alignments Here we focus on using alignment to translate expressions between libraries. However, alignments also allow for a variety of other services such as simultaneous browsing and searching of multiple corpora, aiding premise selection, statistical analogies or refactoring of formal corpora; for more details we refer to [Mül+17].

Overview In Sect. 2, we recap the relevant preliminaries about alignments and the MMT framework. Then we describe the interface theories and our alignments in Sect. 3 and 4, respectively. It is worth noting that these results were not found in that order: we first collected a large set of alignments between the formal libraries, then we constructed interface theories by abstracting over these alignments. In Sect. 5 we apply our design to concrete expression translations. We conclude in Sect. 6 with a vision of a future collaborative effort expanding our work with many more interface theories and alignments.

2 Preliminaries

2.1 Alignments

We speak of *alignment* if the same (or a closely related) concept occurs in different libraries, possibly with slightly different names, notations, or formal definitions. Two aligned symbols thus can be thought to represent "the same" abstract mathematical concept while differing only in implementation details. The notion of alignments is intentionally broad as to cover a wide variety of ways in which two symbols can be seen as "morally the same".

For example, consider untyped and typed equality, e.g. in a set-theoretical and a dependently typed language respectively:

$$\mathsf{eq}_1 : \mathsf{Set} \to \mathsf{Set} \to \mathsf{bool}$$
$$\mathsf{eq}_2 : (T : \mathsf{Type}) \to T \to T \to \mathsf{bool}.$$

Obviously, these two symbols are not easily interchangeable since they don't even have the same type; nevertheless, if we assume either symbol belonging to a distinct formal system, they clearly represent the same mathematical concept (namely *equality*). Furthermore, if we want to translate any expression from one system to the other, we will clearly need to replace occurrences of one kind of equality by the respective other one.

It thus makes sense to think of both equalities as being *aligned*.

For example, the alignment between eq_1 and eq_2 can be given as

$$\mathsf{system1{:}eq}_1\ \mathsf{system2{:}eq}_2\ \mathsf{arguments}=\text{“}(1,2)(2,3)\text{”}\ \mathsf{direction}=\text{“}\mathsf{both}\text{”}$$

For details on alignments we refer to [Mül+17], where alignments are classified, discussed and their implementations in MMT are described in detail.

2.2 The MMT Framework

MMT [RK13; Rab14] is a wide-coverage representation language for formal mathematical knowledge. It can be seen as a triple of the fragment of OMDoc [Koh06] dealing with logical and related knowledge, a rigorous semantics for that fragment, and a mature implementation. OMDoc/MMT is designed to

- avoid a commitment to any particular foundational type system or logic,
- allow for highly modular representations of foundational systems or domain knowledge,

- support interoperability across foundations, tools, and libraries.

That makes it an ideal choice for describing interface theories.

In the sequel, we explain by example the key features of MMT that are relevant for our purposes. Figure 1 shows an example theory in MMT surface syntax.

```
namespace http://mathhub.info/MitM/Foundation ❙

theory Logic =
   bool : type ❙

   true : bool ❙
   false : bool ❙

   ded : bool → type ❙ # ⊢ 1 prec -500 ❙

   eq : {A:type} A → A → bool ❙ # 2 ≐ 3 prec -5
   and : bool → bool → bool ❙ # 1 ∧ 2 prec -110 ❙
   implies : bool → bool → bool ❙ # 1 ⇒ 2 prec -130 ❙
```

Figure 1: An interface theory for logic in MMT

Here, a new theory `Logic` is declared with URI http://mathhub.info/MitM/Foundation?Logic, which is composed of the namespace declared in the first line and the name of the theory. Afterwards, three constants are declared and given a type; namely the type of booleans `bool` of type `type`. The symbol `type` is provided via a logical framework. The constants `true` and `false` are declared to be of type `bool`.

In general, MMT constants have the form `c[: TYPE][= DEF][#NOT]`, though we will not need definitions in this paper. The individual components (type, definition, notation, respectively) are all optional and can be provided in any order.

The symbol `ded` will serve as a function from propositions to the type of their proofs to make use of the Curry-Howard correspondence. It is given the appropriate type `bool` $\to$ `type` and a notation `⊢ 1`. The latter allows for writing `⊢ A` for the type of proofs of a proposition A. Furthermore, the notation is provided with a precedence, to allow for omitting brackets.

The next symbols provide a typed equality and the basic logical connectives. The curly braces denote the dependent function type, providing for example `eq` with the type $\prod_{A:\mathsf{type}} A \to A \to \mathsf{bool}$. The notation $2 \doteq 3$ omits the first argument `A : type`, leaving it implicit and to be inferred by the system.

We can use this theory as one (out of many possible) meta-theory for various interface theories. Figure 2 shows an example of a simple theory of natural numbers, using the theory `Logic` and the symbols declared therein.

Using this modular approach as well as the foundation-independet nature of OMDoc/MMT, the core primitives of various formal systems (such as interactive theorem provers) can be (and in some cases have been, see e.g. [OAF] or [KR16] for the big picture) represented in OMDoc/MMT alongside their libraries by using a formalization of the former as meta-theory for the theories in the latter, making either equally accessible to the system. This is what makes MMT a "meta-system" suitable for our purposes, instead of the $n+1$st competing standard.

```
theory NaturalNumbers : fnd:?Logic =

  NaturalNumbers : type # ℕ

  zero      : ℕ
  successor : ℕ → ℕ          # S 1
  plus      : ℕ → ℕ → ℕ      # 1 + 2
  times     : ℕ → ℕ → ℕ      # 1 · 2
  leq       : ℕ → ℕ → bool   # 1 ≤ 2

  Induction : {P}   ⊢ P zero → ({n} ⊢ P n → ⊢ P (S n)) → ⊢ ∀ P
  Succ_inj  : {n,m} ⊢ (S n) ≐ (S m) → ⊢ n ≐ m

  plus_zero : {n}   ⊢ (n + zero) ≐ n
  plus_succ : {n,m} ⊢ (n + (S m)) ≐ S (n + m)

  times_zero : {n}   ⊢ (n · zero) ≐ zero
  times_succ : {n,m} ⊢ (n · (S m)) ≐ ((n · m) + n)
```

Figure 2: An interface theory for natural numbers with Logic as meta-theory

3 Interface Theories

While alignments have the big advantage that they are cheap to find and implement, they have the disadvantage of not being very expressive; in fact, the definition of alignment itself is (somewhat intentionally) vague[1]. In particular when a translation needs to consider foundational aspects beyond the individual system dialects, alignments alone are insufficient. This is where *interface theories* ([KRSC11; CFK14]) come into play: given different implementations of the same mathematical concept, their interface theory contains only those symbols that are a) common to all implementations and b) necessary to use the concept from the outside – which in practice turn out to be the same thing.

3.1 Example: Natural Numbers

In Mizar [Miz], which is based on Tarski-Grothendieck set theory, the set of natural numbers `NAT` is defined as omega, the set of finite ordinals. Arithmetic operations are defined directly on those.

In contrast, in PVS [ORS92] natural numbers are defined as a specific subtype of the integers, which in turn are a subtype of the rationals etc. up to an abstract type number which serves as a maximal supertype to all number types. The arithmetic operations are inherited from a subtype number_field of number.

These are two fundamentally different approaches to describe and implement an abstract mathematical concept, but for all practical purposes the concept they describe is the same; namely the natural numbers. The interface theory for both variants would thus only contain the symbols that are relevant to the abstract concept itself, independent of their specific implementation – hence, things like the type of naturals, the arithmetic operations and the Peano axioms. The interface theory thus provides everything we need to work with natural numbers, and at the same time everything we know about them independently of the logical foundation or their specific implementation within any given formal system.

However, there is an additional layer of abstraction here, namely that in stating that the natural numbers in Mizar are the finite ordinals we have already ignored the system dialect (in the sense of p.2). This step of abstraction (from the concrete definition using only Mizar-specific symbols) yields another interface

[1] In [Mül+17], we attempt to classify alignments in more detail to contribute to a solution.

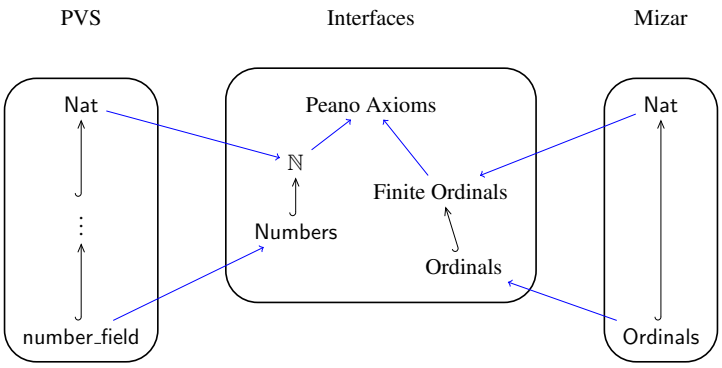


Figure 3: A graph showing different theories for natural numbers

theory for finite ordinals, which in turn can be aligned not just with Mizar natural numbers, but also e.g. with MetaMath [MeMa], which is built on ZFC set theory.

Figure 3 illustrates this situation. Blue arrows point from more detailed theories to their interfaces. The arrows from PVS or Mizar to interfaces merely strip away the system dialects; the arrows within Interfaces abstract away more fundamental differences in definition or implementation.

Consider again Figure 2, a possible interface theory for natural numbers. Note, that symbols such as leq *could* be defined, but don't actually need to be. Since they are only interfaces, all we need is for the symbols to exist.

In fact, the more abstract the interface, the less we *want* to define the symbols – given that there's usually more than one way to define symbols, definitions are just one more thing we might want to abstract away from completely.

The symbols in this interface theory can then be aligned either with symbols in other formal systems directly, or with additional interfaces in between, such as a theory for Peano arithmetic, or the intersection of all inductive subsets of the real numbers, or finite ordinals or any other possible formalization of the natural numbers.

3.2 Additional Interface Theories

The foundation independent nature of MMT allows us to implement interface theories with almost arbitrary levels of detail and for vastly different foundational settings.

We have started a repository of interface theories specifically for translation purposes [Mitc] and also aligned to already existing interfaces (as in the case of arithmetics, see below) in a second and third MathHub repository [Mitd] and [Mitb] extending them when necessary. Crucially, this interface repository contains interface theories for basic type-related symbols like the function type constructors (see Figure 4), that are aligned with the respective symbols in HOL Light and PVS. These symbols are so basic as to be primitive in systems based on type theory, and consequently they occur in the vast majority of expressions. To have these symbols aligned is strictly necessary to get any further use of alignments off the ground.

Here, a *structure* is used to include the theory for simple function types in the theory for dependent function types, while providing definitions for the symbols in terms of the latter. This automatically yields a translation from the simple to the dependent variant.

```
theory TypeSystem : ur:?LF =
  tp : type
  tm : tp → type

theory SimpleFunctionTypes : ?TypeSystem =
  Arrowtp : tp → tp → tp # 1 ⟹ 2
  Applytp : {A,B} tm (A ⟹ B) → tm A → tm B # 3 @ 4 ## 3 4
  Lambdatp : {A,B} (tm A → tm B) → tm (A ⟹ B) # λ 3

theory DependentFunctionTypes : ?TypeSystem =
  Pitp : {A} (tm A → tp) → tp # Π 2
  Applytp : {A,B} tm (Pitp A B) → {x: tm A} tm (B x) # 3 @ 4 ## 3 4
  Lambdatp : {A,B} ({x: tm A} tm (B x)) → tm (Π B) # λ 3
  structure simple : ?SimpleFunctionTypes =
    Arrowtp = [A,B] Pitp A ([x]B)
    Applytp = [A,B][f][a] Applytp A ([x]B) f a
    Lambdatp = [A,B][f] Lambdatp A ([x]B) f
```

Figure 4: Interface theories for type-theoretical foundations

Table 3 shows the total number of alignments we found for PVS, HOL Light and Mizar. The following are some additional examples of mathematical areas covered by the current interface theories:

Calculus contains the following 3 subinterfaces:

- **Limits** currently contains 17 symbols related to sequences and limits, including `metric_space` and `complete` (metric spaces and their completeness).
- **Differentiation** currently contains 4 symbols, namely differentiability in a point and on a set and the derivative in a point and as a function
- **Integration** currently contains 6 symbols, namely integrability and the integral over a set for Riemann, Lebesgue and Gauge-integration.

Arithmetics is an already existing interface theory from [Mitd]. It contains the interfaces for below number arithmetics (each split into two interfaces for the basic number type definitions and the arithmetics on them).

- **Complex Numbers** currently contains 11 symbols for complex numbers aligned to their counterparts in HOL Light, PVS and Mizar. Besides the usual arithmetic operations similar to `NaturalNumbers`, it contains `i` (the imaginary unit), `abs` (the modulus of a complex number) and `Re`, `Im` (the real and imaginary parts of a complex number).
- **Integers** currently contains 9 symbols for the usual arithmetic operations on integers and for comparison between two integers.
- **Natural Numbers** currently contains 21 symbols and is already described above.
- **Real Numbers** currently contains 15 symbols, again very similar in nature to the other number spaces.

Lists currently contains the 13 most important symbols for lists, including `head`, `tail`, `concat`, `length` and `filter` (filter a list using another list) as well as some auxiliary definitions. There are no lists in Mizar, instead finite sequences are used. These however deserve their own interface.

Logic is an already existing interface in the [Mitb] repository. It contains 9 symbols for boolean algebra that are all perfectly aligned to HOL Light, PVS and Mizar the like and sometimes also to Coq.

Sets is again an already existing interface theory from [Mitd] split into many subtheories. Currently, 28 of the contained symbols have been aligned. sets contains symbols for typed sets as in a type theoretical setting, including axioms and theorems. Here we have the most alignments so far. It also contains the following two interfaces:

Relations currently contains 23 symbols for alignments to relations and their properties, including orders.

Functions currently contains 7 symbols for alignments to functions and their relations, that are not already contained in relations.

Topology currently contains 25 symbols for both general topological spaces as well as the standard topology on $\mathbb{R}^n$ specifically. Since this yields additional difficulties, it will be examined in more detail in the next section.

As additional examples, the interface theories for limits and sets can be found in Appendix A, Figure 7 and Figure 8.

4 Alignments

4.1 Sample Alignments

We manually combed through libraries of HOL Light, PVS, Mizar and Coq to find alignments. Specifically, we picked the mathematical areas of numbers, sets (as well as lists), abstract algebra, calculus, combinatorics, logic, topology, and graphs as a sample. This produced around 900 declarations overall, from which we constructed the interface theories presented in Sect. 3.

Interface	PVS (Standard)	HOL Light (Standard)	Mizar (Standard)	Coq (Standard)
nat_lit	naturalnumbers?naturalnumber	nums?nums	ORDINAL1?modenot.6	Coq.Init.Datatypes?nat
succ	naturalnumbers?succ	nums?SUC	ORDINAL1?func.1	Coq.Init.Nat?succ
addition	number_fields?+	arith?ADD	ORDINAL2?func.10	Coq.Init.Nat?add
multiplication	number_fields?*	arith?MULT	ORDINAL2?func.11	Coq.Init.Nat?mul
lethan	number_fields?<=	arith?<=	XXREAL_0?pred.1	Coq.Init.Nat?leb

Table 1: Alignments to the interface theory NaturalNumbers (libraries in brackets)

Interface	PVS (NASA[2])	HOL Light (Standard)	Mizar (Standard)	Coq (coq-topology[3])
topology	topology_prelim?topology	topology?topology	PRE_TOPC?modenot.1	TopologicalSpaces?TopologicalSpace
open	topology?open?	topology?open_in	PRE_TOPC?attr.3	TopologicalSpaces?open
closed	topology?closed?	topology?closed_in	PRE_TOPC?attr.4	TopologicalSpaces?closed
interior	topology?interior	topology?interior	TOPS_1?func.1	InteriorsClosures?interior
closure	topology?Cl	topology?closure	PRE_TOPC?func.2	InteriorsClosures?closure

Table 2: Alignments to the interface theory Topology (libraries in brackets)

For example, Table 1 and 2 show some of these alignments for the interface theories for natural numbers from Figure 3 and topology. The URIs for PVS consist of the name of the containing theory in PVS (the alignments from the first table are to PVS's prelude, the alignments from the second are to the NASA library) and the name of the symbol within the interface, separated by a question mark[4]. The ? appearing at the end of some of the URIs are actually a part of the name of the symbol in PVS. The URIs

for HOL Light (prelude) and Mizar (mll) consist of the filename and the symbol name within the file.

As mentioned earlier, alignments in topology pose some additional difficulties. Firstly, HOL Light defines a topology on some *subset* of the universal set of the type, whereas PVS defines it on the universal set of the type directly. Thus, the alignment from HOL Light to the interface theory is unidirectional. Secondly, Mizar does not define the notion of a topology, but instead the notion of a topological space. Therefore, we align them to two different symbols in the interface (`topology` and `topological_space`) and define `topological_space` based on `topology` , so that MMT can still translate expressions based of these two definitions.

The set of all alignments we found can be inspected at https://gl.mathhub.info/alignments/Public/tree/master/manual.

4.2 Alignment Directions

For translation purposes, we distinguish among three kinds of alignments (which are different from the categories in [Mül+17]) based on the possible directions of translations between the concepts in the interface theory and the prover library.

Bidirectional Alignments This includes perfect alignments. For example, the translation between the definitions of set union in the interface theory and PVS library is bidirectional:

Interface `union : {A} set A → set A → set A # 2 ∪ 3`

PVS `union(a, b): set = x | member(x, a) OR member(x, b)`

Unidirectional Alignments This includes alignments up to totality of functions, alignments up to certain arguments and alignments up to associativity. For example, in PVS the operator `+` is used universally for all number fields ($\mathbb{N}$, $\mathbb{R}$, $\mathbb{C}$), but in the interface theory `plus` is defined for each type. Thus we can only translate from the interface theory to PVS.

Other Alignments Due to the limitation of MMT's current implementations, there are some alignments which cannot be used at the moment but are potentially directional. For example, the `>` operator in Mizar is not explicitly declared, but instead defined as the so-called `antonym` of the `<=` operator. It is redirected to the `<=` operator whenever used:

```
antonym a > b for a <= b;
```

However, since MMT cannot handle alignments up to negation, this cannot be used for translation yet.

4.3 Causes for Imperfect Alignments

The most common causes for imperfect alignments are subtyping and partiality.

[2] See http://github.com/nasa/pvslib

[3] See http://github.com/coq-contribs/topology

[4] The general structure of MMT URIs is <Namespace>?<Theory name>?<Symbol name>, see e.g. [RK13]

Topic	HOL Light	PVS	Mizar	Coq
Algebra	0/0	18/1	17/0	14/0
Calculus	15/0	14/0	16/0	5/15
Categories	0/0	0/0	9/1	5/0
Combinatorics	24/0	15/0	1/0	1/0
Complex Numbers	9/2	4/6	7/2	11/2
Graphs	5/5	17/0	20/0	7/2
Integers	10/0	0/0	5/2	47/3
Lists	16/0	9/0	8/0	36/2
Logic	7/0	7/5	7/0	24/1
Natural Numbers	19/0	8/10	9/0	34/1
Polynomials	4/0	1/0	7/0	0/0
Rational Numbers	0/14	2/11	0/10	14/3
Real Numbers	13/2	3/10	7/4	12/2
Relations	4/0	16/5	18/3	1/12
Sets	23/0	28/0	18/0	19/0
Topology	15/0	10/0	9/0	17/1
Vectors	13/0	7/0	15/0	0/0
Sum	**177/23**	**159/48**	**173/22**	**240/42**

Table 3: Number of bidirectional/unidirectional alignments per library

Subtyping In the PVS library, the arithmetic operations on all the number fields are defined on a common supertype `numfields`. Therefore, a translation from PVS to the other two languages may not be viable.

Partiality The result of division by zero in the libraries of HOL Light and Mizar is defined as zero; in PVS, however, the divisor must be nonzero. Therefore, certain theorems in HOL Light and Mizar involving division no longer hold in PVS, and the translation is unidirectional.

In order to translate an expression from one library to another, the concepts in the expression must at least exist in both libraries. This creates the need to inspect the intersection of the concepts in these libraries. Table 4 gives an overview of the library intersection for various interface theories.

5 An Implementation of Alignment-based Translations

We can use alignments to translate expressions between different representations of the same mathematical concept. We can do so by simply substituting all occurrences of a symbol a_1 by the aligned symbol a_2 if translation is possible in this direction. For example, consider the expression `member(a,A)` in PVS, stating that $a \in A$ for some set A of type T. In OMDoc, this corresponds to the term (using pseudo-URIs)

`PVS?apply(PVS/sets?member,T,a,A).`

Since `member` is aligned with `IN` in HOL Light, we can simply substitute one by the other. Furthermore, the symbol `PVS?apply` representing function application in PVS can be assumed to be aligned with HOL Light's function application, yielding the term `HOLLight?apply(HOLLight/Sets?IN,T,a,A)`, which is the HOL Light expression `aINA` in OMDoc.

The additional components of alignments allow for stating additional translation instructions such as switching, adding or omitting arguments.

Topic	1 System	2 Systems	3 Systems	4 Systems
Algebra	17	9	5	0
Calculus	35	7	8	0
Categories	4	5	0	0
Combinatorics	25	6	0	1
Complex Numbers	10	5	3	3
Graphs	72	6	3	0
Integers	52	2	7	0
Lists	28	8	9	0
Logic	18	0	2	5
Natural Numbers	53	2	10	2
Polynomials	12	0	0	0
Rational Numbers	11	4	2	7
Real Numbers	9	3	5	5
Relations	21	15	4	0
Sets	56	10	9	10
Topology	62	2	8	0
Vectors	25	5	0	0
Sum	**510**	**91**	**79**	**33**

Table 4: Number of concepts found in exactly one, two, three or four systems

For larger differences in implementation for which alignments are insufficient, we instead use interface theories for the (more or less) specific implementation variants that abstract away the system dialects. This allows us to implement arbitrary elaborate translation mechanisms on a generic level, without needing to care about the system-specific details.

Theory morphisms can connect and translate from more detailed implementations to more abstract ones, e.g. going from finite ordinals to a first-order theory of natural numbers. Other translation mechanisms such as one from a set-theoretical to a type-theoretical setting can be realized generically on interface theories and thus can be leveraged by multiple systems. The LATIN library [Cod+11; LATIN] already provides a variety of logics, type theories, set theories and translations between them in MMT that can be used for these purposes.

Figure 5 shows possible relations on the theory level for the natural numbers example above: red lines stand for alignments, hooked arrows for theory inclusions, straight arrows for views and dotted arrows for other possible translations.

Given the various arrows, expression translation reduces to a simple graph search on the symbols (or, depending, on larger subexpressions) of the input.

While we only translate single expressions instead of whole proofs, the former is a necessary first step for the latter. Furthermore, translating the expressions occuring in a proof yields a "proof sketch" in the target system, which – we conjecture – can often be enough to complete the proof there automatically.

We have implemented a prototypical expression translation in MMT. It currently uses ≈ 400 alignments for translation purposes (out of ≈ 1400 in total) between HOLLight, PVS, Mizar and Coq (even though there is no OMDoc import from Coq as of yet). The translation mechanism will be exposed via the MMT query language [Rab12]. Besides alignments, it uses views, MMT-structures and arbitrary other translations implemented in Scala and provided by MMT plugins. Furthermore, it allows for grouping alignments so they are always used in conjunction, and prioritizing certain translations over others.

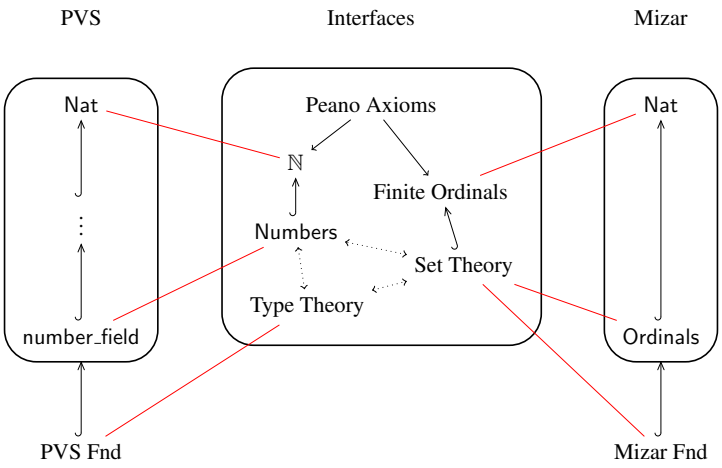


Figure 5: A graph showing different translation paths between theories

Crucially, it returns partial translations when no full translation to a target library can be found. These can be used for finding new alignments automatically by the techniques described in the introduction.

We furthermore have two libraries for interface theories; MitM/interfaces [Mitc] specifically for expression translation and, more generally MitM [Mita]. The latter is additionally used in other projects for simlar purposes ([Koh+17b] and [ODKa]).

Figure 6 shows a small part of the theory graph of the MitM libraries. The full MitM theory graph can be explored at https://mathhub.info/mh/mmt/graphs/tgview.html?type=archivegraph&graphdata=MitM

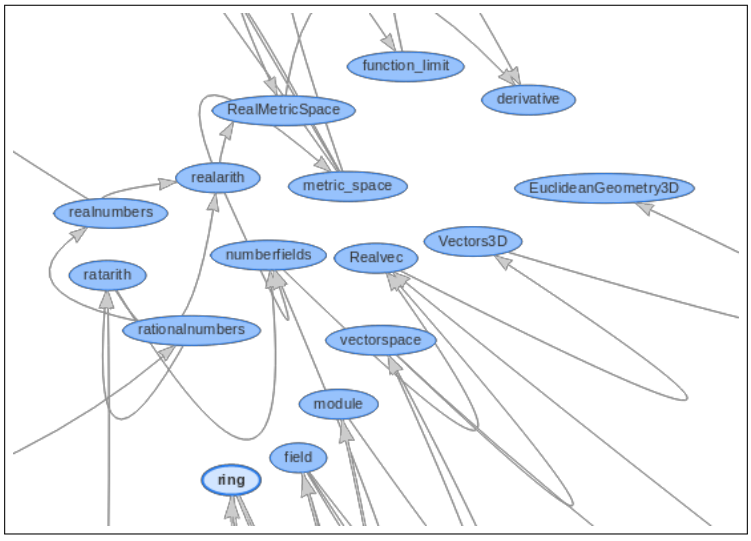


Figure 6: A Small Part of the MitM Theory Graph

The above can be (and has been) used to – exemplary – translate the expressions listed in Table 5. The curly braces denote the context for the variables used.

Note that even something like the application of a function entails a translation from function appli-

cation in HOLLight to function application in PVS, since they belong to their respective system dialects. Furthermore, the translation from simple function types in HOLLight to the dependent Π-type in PVS is actually done on the level of interface theories, where simple function types are defined as a special case of dependent function types for this exact purpose. Unlike the latter two, the first example uses only primitive symbols in both systems that are not part of any external library. Meta-variables have been left out in the third example for brevity.

Furthermore, an argument alignment was used in the third example, that switches the two (non-implicit) arguments – namely the predicate and the list. Whereas this usually trips up automated translation processes or needs to be manually implemented, in our case this is as easy as adding (in this case) the key-value-pair `arguments` = "(2,3)(3,2)" to the alignment.

HOL Light	PVS
{A:holtype, P:term A⟹bool, a:term A}⊢P(a)	{A:tp,P:expr $\Pi_{a:A}$boolean,a:expr A}⊢P(a)
{T:holtype,a:term T,A:term T⟹bool}a IN A	{T:tp, a:expr T, A:expr T⟹boolean}member(a,A)
FILTER (Abs x:bool. x) c :: b :: a :: NIL	filter (c :: b :: a :: null) (λx : boolean. x)

Table 5: Three Expressions Translated

6 Conclusion

We presented a case study that demonstrates that the approach of alignment-based translation of expressions between theorem prover libraries is feasible. Concretely, we gave interface theories for five representative mathematical domains along several hundred alignments to three major libraries.

Our alignments are publicly available at [AP]. Our representations of libraries in MMT (including HOLLight, Mizar and PVS) can be found on MathHub [MH]. We currently pursue additional imports from IMPS [FGT93], TPS and Coq.

We see this paper as an initial, prototypical example of a community-wide effort to build a large library of interface theories and alignments. We envision an interface library that subsumes all formal knowledge formalized in major theorem provers. In the long run, new formalizations in any system should always be coupled with an interface theory that is uploaded to this central library. (If successful, this agenda will eventually lead to an integration problem when multiple competing interface theories exist for the same domain. But, while non-trivial, that problem is much simpler than the integration problem between libraries.)

Therefore, we call on the community to expand our set of interface theories and alignments, both in future publications and through pull requests to the above-mentioned git repository. If an interface theory is already known or (as in our case) roughly known, finding more alignments is relatively easy. For example, the alignments presented here were found by two strong undergraduate students in a total of around 230 hours with relatively little supervision. Reporting the found alignments and parallelizing the work is easy using the standardized format presented in [Kal+16]. Therefore, we expect it will be easy to scale the manual approach up. In fact, finding alignments can be a great exercise for students to familiarize themselves with a library.

References

[AP] *Public Repository for Alignments*. URL: https://gl.mathhub.info/alignments/Public.

[BCH12] M. Boespflug, Q. Carbonneaux, and O. Hermant. "The λΠ-calculus modulo as a universal proof language". In: *Proceedings of PxTP2012: Proof Exchange for Theorem Proving*. Ed. by D. Pichardie and T. Weber. 2012, pp. 28–43.

[CFK14] Jacques Carette, William Farmer, and Michael Kohlhase. "Realms: A Structure for Consolidating Knowledge about Mathematical Theories". In: *Intelligent Computer Mathematics 2014*. Ed. by Stephan Watt et al. LNCS 8543. MKM Best-Paper-Award. Springer, 2014, pp. 252–266. ISBN: 978-3-319-08433-6. URL: http://kwarc.info/kohlhase/papers/cicm14-realms.pdf.

[Cic] *Intelligent Computer Mathematics*. LNAI. in press. Springer, 2017.

[CICM1616] Paul-Olivier Dehaye et al. "Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach". In: *Intelligent Computer Mathematics 2016*. Ed. by Michael Kohlhase et al. LNAI 9791. Springer, 2016. URL: https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/CICM2016/published.pdf.

[Cod+11] Mihai Codescu et al. "Project Abstract: Logic Atlas and Integrator (LATIN)". In: *Intelligent Computer Mathematics*. Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 289–291. ISBN: 978-3-642-22672-4. URL: https://kwarc.info/people/frabe/Research/CHKMR_latinabs_11.pdf.

[Dav+11] James Davenport et al., eds. *Intelligent Computer Mathematics*. LNAI 6824. Springer Verlag, 2011. ISBN: 978-3-642-22672-4.

[FGT93] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. "IMPS: An Interactive Mathematical Proof System". In: *Journal of Automated Reasoning* 11.2 (Oct. 1993), pp. 213–248.

[GK14] Thibault Gauthier and Cezary Kaliszyk. "Matching concepts across HOL libraries". In: *CICM*. Ed. by Stephen Watt et al. Vol. 8543. LNCS. Springer Verlag, 2014, pp. 267–281. DOI: 10.1007/978-3-319-08434-3_20.

[IKR11] Mihnea Iancu, Michael Kohlhase, and Florian Rabe. *Translating the Mizar Mathematical Library into OMDoc format*. KWARC Report. Jacobs University Bremen, 2011. URL: http://uniformal.github.io/doc/applications/LATIN/docs/Mizar2OMDoc-Report.pdf.

[Kal+16] Cezary Kaliszyk et al. "Aligning the Libraries of Formal Mathematical Systems". submitted to CPP 2016. 2016. URL: http://kwarc.info/kohlhase/submit/alignments16.pdf.

[Koh+17a] Michael Kohlhase et al. "Making PVS Accessible to Generic Services by Interpretation in a Universal Format". In: accepted at ITP 2017. 2017. URL: http://kwarc.info/kohlhase/submit/itp17-pvs.pdf.

[Koh+17b] Michael Kohlhase et al. "Mathematical models as research data via flexiformal theory graphs". In: *Intelligent Computer Mathematics (CICM) 2017*. LNAI. in press. Springer, 2017. URL: http://kwarc.info/kohlhase/papers/cicm17-models.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.

[KR14] Cezary Kaliszyk and Florian Rabe. "Towards Knowledge Management for HOL Light". In: *Intelligent Computer Mathematics 2014*. Ed. by Stephan Watt et al. LNCS 8543. Springer, 2014, pp. 357–372. ISBN: 978-3-319-08433-6. URL: http://kwarc.info/frabe/Research/KR_hollight_14.pdf.

[KR16] Michael Kohlhase and Florian Rabe. "QED Reloaded: Towards a Pluralistic Formal Library of Mathematical Knowledge". In: *Journal of Formalized Reasoning* 9.1 (2016), pp. 201–234. URL: http://jfr.unibo.it/article/download/4570/5733.

[KRSC11] Michael Kohlhase, Florian Rabe, and Claudio Sacerdoti Coen. "A Foundational View on Integration Problems". In: *Intelligent Computer Mathematics*. Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 107–122. ISBN: 978-3-642-22672-4. URL: http://kwarc.info/kohlhase/papers/cicm11-integration.pdf.

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

[MeMa] *Metamath Home page*. URL: http://us.metamath.org.

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

[Mita] *MitM: The Math-in-the-Middle Ontology*. URL: https://mathhub.info/MitM (visited on 02/05/2017).

[Mitb] *MitM/Foundation*. URL: https://gl.mathhub.info/MitM/Foundation (visited on 09/15/2017).

[Mitc] *MitM/Interfaces*. URL: https://gl.mathhub.info/MitM/interfaces (visited on 06/21/2017).

[Mitd] *MitM/smglom*. URL: https://gl.mathhub.info/MitM/smglom (visited on 09/14/2017).

[Miz] *Mizar*. http://www.mizar.org. 1973–2006. URL: http://www.mizar.org.

[MK15] Dennis Müller and Michael Kohlhase. "Understanding Mathematical Theory Formation via Theory Intersections in MMT". 2015. URL: http://cicm-conference.org/2015/fm4m/FMM_2015_paper_2.pdf.

[Mül+17] Dennis Müller et al. "Classification of Alignments between Concepts of Formal Mathematical Systems". In: *Intelligent Computer Mathematics (CICM) 2017*. LNAI. in press. Springer, 2017. URL: http://kwarc.info/kohlhase/submit/cicm17-alignments.pdf.

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

[ODKa] *Open Digital Research Environment Toolkit for the Advancement of Mathematics*. Project Proposal. URL: https://github.com/OpenDreamKit/OpenDreamKit/raw/master/Proposal/proposal-www.pdf (visited on 09/01/2015).

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

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. "PVS: A Prototype Verification System". In: *Proceedings of the 11th Conference on Automated Deduction*. Ed. by D. Kapur. LNCS 607. Saratoga Springs, NY, USA: Springer Verlag, 1992, pp. 748–752.

[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. ISBN: 978-3-540-37188-5. DOI: 10.1007/11814771_27.

[Rab12] Florian Rabe. "A Query Language for Formal Mathematical Libraries". In: *Intelligent Computer Mathematics*. Ed. by Johan Jeuring et al. LNAI 7362. Berlin and Heidelberg: Springer Verlag, 2012, pp. 142–157. ISBN: 978-3-642-31373-8. arXiv: 1204.4685 [cs.LO].

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

[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.

[Wat+14] Stephan Watt et al., eds. *Intelligent Computer Mathematics*. LNCS 8543. Springer, 2014. ISBN: 978-3-319-08433-6.

Appendices

A More Examples of Interface Theories

```
theory Analysis : fnd:?Logic =
    include arith:?natarith
    include ?Topology
    include ?Sets
    sequence : type → type = [T] ℕ → T
    Rn : type # ℝn
    metric_space : type # ℝ

    convergent_to : {A} sequence A → A → bool # 2 --> 3
    convergent : {A} sequence A → bool # convergent 2
    limit : sequence ℝ → ℝ # limit 2
    monotone : sequence ℝ → bool # monotone 1
    cauchy_Rn : sequence ℝ → bool # cauchy 1
    complete_Rn : set ℝn → bool # complete 1
    uniformly_continuous_Rn : (set ℝn → set ℝn) → bool # uniformly_continuous 1

    cauchy : {A} metric_space A → sequence A → bool # cauchy 3
    convergent_metr : {A} metric_space A → sequence A → bool # convergent 2 3
    complete : {A} metric_space A → set A → bool # complete 3
    uniformly_continuous : {A} metric_space A → {B} topological_space B → (set A → set B) → bool # uniformly_continuous 5
```

Figure 7: An interface theory for analysis

```
theory Sets : fnd:?Logic =
    set : type → type = [T] T → bool
    member : {A} A → set A → bool # 2 in 3
    emptyset : {A} set A # ∅ %l1
    union : {A} set A → set A → set A # 2 ∪ 3
    universe : {A} set A # universe %l1
    complement : {A} set A → set A # complement 2
    UNION : {A} set (set A) → set A # UNION 2
    intersection : {A} set A → set A → set A # 2 ∩ 3
    INTERSECTION : {A} set (set A) → set A # INTERSECTION 2
    singleton : {A} A → set A # singleton 2
    difference : {A} set A → set A → set A # 2 \ 3
    symmetric_difference : {A} set A → set A → set A # 2 \+\ 3
    is_subset : {A} set A → set A → bool # 2 ⊆ 3
    is_proper_subset : {A} set A → set A → bool # 2 ⊂ 3
    is_disjoint : {A} set A → set A → bool # disjoint 2 3
    is_singleton : {A} set A → bool # singleton 2
    powerset : {A} set A → set (set A) # powerset 2
    extensionality : {A, s : set A, t : set A} ⊢ ∀ [a : A] (a in s ⇔ a in t) ⇒ s ≐ t
    universe_member : {A} ⊢ ∀ [a : A] (a in universe)
    subset_transitive : {A} ⊢ ∀ [s : set A] ∀ [t : set A] ∀ [u : set A] (s ⊆ t ∧ t ⊆ u) ⇒ s ⊆ u
    subset_reflexive : {A} ⊢ ∀ [s : set A] s ⊆ s
    subset_antisymmetric : {A} ⊢ ∀ [s : set A] ∀ [t : set A] (s ⊆ t ∧ t ⊆ s) ⇒ s ≐ t
    subset_emptyset : {A} ⊢ ∀ [s : set A] ∅ ⊆ s
    emptyset_subset : {A} ⊢ ∀ [s : set A] ( s ⊆ ∅) ⇒ s ≐ ∅
    proper_subset_transitive : {A} ⊢ ∀ [s : set A] ∀ [t : set A] ∀ [u : set A] (s ⊂ t ∧ t ⊂ u) ⇒ s ⊂ u
    proper_subset_irreflexive : {A} ⊢ ∀ [s : set A] ¬ (s ⊂ s)
```

Figure 8: An interface theory for typed sets