Classification of Alignments between Concepts of Formal Mathematical Systems

Dennis Müller[1], Thibault Gauthier[2], Cezary Kaliszyk[2],
Michael Kohlhase[1], Florian Rabe[3]

[1] FAU Erlangen-Nürnberg
[2] University of Innsbruck
[3] Jacobs University

Abstract. Mathematical knowledge is publicly available in dozens of different formats and languages, ranging from informal (e.g. Wikipedia) to formal corpora (e.g., Mizar). Despite an enormous amount of overlap between these corpora, only few machine-actionable connections exist. We speak of *alignment* if the same concept occurs in different libraries, possibly with slightly different names, notations, or formal definitions. Leveraging these alignments creates a huge potential for knowledge sharing and transfer, e.g., integrating theorem provers or reusing services across systems. Notably, even imperfect alignments, i.e. concepts that are *very similar* rather than identical, can often play very important roles. Specifically, in machine learning techniques for theorem proving and in automation techniques that use these, they allow learning-reasoning based automation for theorem provers to take inspiration from proofs from different formal proof libraries or semi-formal libraries even if the latter is based on a different mathematical foundation. We present a classification of alignments and design a simple format for describing alignments, as well as an infrastructure for sharing them. We propose these as a centralized standard for the community. Finally, we present an initial collection of ≈ 12000 alignments from the different kinds of mathematical corpora, including proof assistant libraries and semi-formal corpora as a public resource.

1 Introduction

Motivation The sciences are increasingly collecting and curating their knowledge systematically in machine-processable corpora. For example, in biology many important corpora take the form of ontologies, e.g., as collected on BioPortal. These corpora typically overlap substantially, and much recent work has focused on integrating them. A central problem here is to find *alignments*: pairs (a_1, a_2) of identifiers from different corpora that describe the same concept, giving rise to *ontology matching* [ESC07].

In the certification of programs and proofs, the ontology matching problem is most apparent when trying to use multiple reasoning systems together. For example, Wiedijk [Wie06] explored a single theorem (and its proof) across 17 proof assistants implicitly generating alignments between the concepts present in the theorem's statement and proof. The Why3 system [BFMP11] maintains a set of translations into different reasoning systems for discharging proof obligations. Each translation must manually code individual alignments of elementary concepts such as integers or lists in order to fully utilize the respective system's automation potential. But automating the generation and use of alignments, which would be necessary to scale up such efforts, is challenging because the knowledge involves rigorous notations, definitions, and properties, which leads to very diverse corpora with complex alignment options. This makes it very difficult to determine if an alignment is *perfect* (we will attempt to define this notion in the next section), or to predict whether an *imperfect* alignment will work just as well or not at all.

Alignment use cases Many practical services are enabled by alignments:

- Simultaneous browsing of multiple corpora. This is already enabled (so far for a limited number of corpora) by our system presented in Section 4.
- Imperfect alignments can be used to search for a single query expression in multiple corpora at once. This has been demonstrated in the Whelp search engine [AGC$^+$04] where Coq and Matita shared the URI syntax with the basic Calculus of Constructions constants aligned, as well as by the MathWebSearch engine [KR14].
- Statistical analogies extracted from large formal libraries combined with imperfect alignments can be used to create new conjectures and thus to automatically explore a logical corpus [GKU16]. This complements the more classical conjecturing and theory exploration mechanisms.
- Automated reasoning services can make use of alignments to provide more precise proof recommendations. The quality of the HOL(y)Hammer proof advice for HOL Light can be improved from 30% to 40% by using the imperfect alignments to HOL4 [GK15].
- Translations between systems. [KK13] uses more than 70 manually discovered alignments between HOL Light and Isabelle/HOL to obtain translated theorems that talk about target system constants and types. Note, that it is not necessary for the translation for the definitions of the concepts to be the same. It is enough if the same properties are provable or if they yield the same computational behavior. Consider the real numbers: In some HOL proof assistants they are defined using Cauchy sequences, while others use Dedekind cuts. The two structures share all the relevant real number properties. However, they disagree with respect to irrelevant properties, e.g., in the construction of the former usually a canonical Cauchy sequence for each real number is introduced. Despite this minor difference, we can use such an alignment in a logical translation [KK13].
- Refactoring of proof assistant corpora. Aligning concepts across versions of the same proof corpus combined with statement normalization and consistent name hashing allowed discovering 39 symbols with equivalent definitions [KU15] in the Flyspeck development [H$^+$15].

Automatic search for alignments Finding alignments, preferably automatically, has proved extremely difficult in general. There are three reasons for this: the conceptual differences between logical corpora found in proof assistants, computational corpora containing algorithms from computer algebra systems, narrative corpora that consists of semi-formal descriptions from wiki-related tools; the diversity of the underlying formal languages and tools; and the differences between the organization of the knowledge in the corpora.

Recently, the second and third authors have developed heuristic methods for automatically finding alignments [GK14] targeted at integrating logical corpora, which we integrate into our developments in Section 2. Independently, Deyan Ginev built a library [GC14] of about 50,000 alignments between narrative corpora including Wikipedia, Wolfram Mathworld, PlanetMath and the SMGloM semantic multilingual glossary for mathematics. For this, the NNexus system indexes the corpora and applies clustering algorithms to discover concepts.

Related Work Alignments between computational corpora occur in bridges between the run time systems of programming languages. Alignments between logical and computational corpora are used in proof assistants with code generation such as Isabelle [WPN08] and Coq [Coq15]. Here functions defined in the logic are aligned with their implementations in the programming language in order to generate fast executable code from formalizations.

The dominant methods for integrating logical corpora so far have focused on truth-preserving translations between the underlying knowledge representation languages. For example, [KS10] translates from Isabelle/HOL to Isabelle/ZF. [KW10] translates from HOL Light to Coq, [OS06] to Isabelle/HOL, and [NSM01] to Nuprl. Older versions of Matita [ACTZ06] were able to read Coq compiled theory files. [CHK+11] build a library of translations between different logics.

However, most translations are not alignment-aware, i.e., it is not guaranteed that a_1 will be translated to a_2 even if the alignment is known. This is because a_1 and a_2 may be subtly incompatible so that a direct translation may even lead to inconsistency or ill-typed results. [OS06] was — to the authors knowledge — the first that could be parametrized by a set of alignments. The OpenTheory framework [Hur09] provides a number of higher-order logic concept alignments. In [KR16], the fourth and fifth author discuss the corpus integration problem and conclude that alignments are of utmost practical importance. Indeed, corpus integration can succeed with only alignment data even if no logic translation is possible. Conversely, logic translations contribute little to corpus integration without alignment data.

Contribution and Overview Our contribution is three-fold.

First, we present a phenomenological study of alignments between proof assistant corpora, as well as with mathematical corpora in Section 2. We show a number of imperfect alignments and show how this can be used to benefit knowledge transfer. Second, we propose a standard for storing and sharing alignments (see Section 4), we cover the central ingredient – global identifiers based on MMT URIs [RK13] – in Section 3. Every symbol is assigned a unique way to access it across corpora and across logics. The URIs are used both in the system and to give several examples from logical and computational corpora in [MGK+17].

Most corpora are developed and maintained by separate, often disjoint communities. That makes it difficult for researchers to utilize alignments because no public resource exists for jointly building a large collection of alignments. Therefore we have started such a resource in form of a central repository as our third contribution — it is public, and we invite all researchers to contribute their alignments. We seeded our repository with the alignment sets mentioned above. Moreover, we are hosting a web-server that allows for conveniently querying for all symbols aligned with a given symbol, currently including ≈ 12000 alignments between proof assistant libraries and 22 alignments to semi-formal corpora (transitive closure not included). We describe this standard and infrastructure in Section 4.

2 Types of Alignments

Let us assume two corpora C_1, C_2 with underlying foundational logics F_1, F_2. We examine examples for how two concepts a_i from C_i can be aligned. Importantly, we allow for the case where a_1 and a_2 represent the same abstract mathematical concept without there being a direct, rigorous translation between them.

The types of alignments in this section are purely phenomenological in nature: they exemplify the difficulty of the problem and provide benchmarks for rigorous definitions. While some types are relatively straightforward, others are so difficult that giving a rigorous definitions remains an open problem. This is because alignments ideally legitimize translations from F_1 to F_2 that replace a_1 with a_2. But in many situations these translations, while possible in principle, are much more difficult than simply replacing one symbol with another. The alignment types below are roughly ordered by increasing difficulty of this translation.

Perfect Alignment If a_1 and a_2 are logically equivalent (modulo a translation φ between F_1 and F_2 that is fixed in the context), we speak of a perfect alignment. More precisely, all formal properties (type, definition, axioms) of a_1 carry over to a_2 and vice versa. Typical examples are primitive types and their associated operations. Consider:

$$\mathtt{Nat}_1 : \mathtt{Type} \qquad \mathtt{Nat}_2 : \mathtt{Type}$$

then translations between C_1 and C_2 can simply interchange a_1 and a_2.

The above example is deceptively simple for two reasons. Firstly, it hides the problem that F_1 and F_2 do not necessarily share the symbol `Type`. Therefore, we need to assume that there are symbols $\mathtt{Type}_1$ and $\mathtt{Type}_2$, which have been already aligned (perfectly). Such alignments are crucial for all fundamental constructors that occur in the types and characteristic theorems of the symbols we want to align such as `Type`, $\to$, `bool`, $\wedge$, etc. These alignments can be handled with the same methodology as discussed here. Therefore, here and below, we assume we have such alignments and simply use the same fundamental constructors for F_1 and F_2.

Secondly, it ignores that we usually want (and can reasonably expect) only certain formal properties to carry over, namely those in the *interface theory* in the sense of [KR16] — i.e. those properties that are still meaningful after abstracting away from the specific foundational logics F_i. For example, in [MGK+17] we give many perfect alignments between symbols that use different but equivalent definitions.

Alignment up to Argument Order Two function symbols can be perfectly aligned except that their arguments must be reordered when translating.

The most common example is function composition, whose arguments may be given in application order $(f \circ g)$ or in diagram order $(f; g)$. Another example is given

$$\begin{aligned}&\texttt{contains}_1 : (T : \texttt{Type}) \to \texttt{SubSet}\, T \to T \to \texttt{bool}\\&\texttt{in}_2 : (T : \texttt{Type}) \to \texttt{T} \to \texttt{SubSet}\, T \to \texttt{bool}\end{aligned}$$

Here the expressions $\texttt{contains}_1(T, A, x)$ and $\texttt{in}_2(T, x, A)$ can be translated to each other.

Alignment up to Determined Arguments The perfect alignment of two function symbols may be broken because they have different types even though they agree in most of their properties. This often occurs when F_1 uses a more fine-granular type system than F_2, which requires additional arguments.

Examples are untyped and typed (polymorphic, homogeneous) equality: The former is binary, while the latter is ternary

$$\begin{aligned}&\texttt{eq}_1 : \texttt{Set} \to \texttt{Set} \to \texttt{bool}\\&\texttt{eq}_2 : (T : \texttt{Type}) \to T \to T \to \texttt{bool}\end{aligned}.$$

The types can be aligned, if we apply $\varphi(\texttt{Set})$ to $\texttt{eq}_2$. Similar examples arise between simply- and dependently-typed foundations, where symbols in the latter take additional arguments.

These additional arguments are uniquely determined by the values of the other arguments, and a translation from C_1 to C_2 can drop them, whereas the reverse translations must infer them – but F_1 usually has functionality for that (e.g. the type parameter of polymorphic equality is usually uniquely determined).

The additional arguments can also be proofs, used for example to represent partial functions as total functions, such as a binary and a ternary division operator

$$\begin{aligned}&\texttt{div}_1 : \texttt{Real} \to \texttt{Real} \to \texttt{Real}\\&\texttt{div}_2 : \texttt{Real} \to (d : \texttt{Real}) \to \vdash d \neq 0 \to \texttt{Real}\end{aligned}$$

Here inferring the third argument is undecidable in general, and it is unique only in the presence of proof irrelevance.

Alignment up to Totality of Functions The functions a_1 and a_2 can be aligned everywhere where both are defined. This often happens since it is often convenient to represent partial functions as total ones by assigning values to all arguments. The most common example is division. $\mathtt{div}_1$ might both have the type $\mathtt{Real} \to \mathtt{Real} \to \mathtt{Real}$ with $x\,\mathtt{div}_1\,0$ undefined and $x\,\mathtt{div}_2\,0 = 0$.

Here a translation from C_1 to C_2 can always replace $\mathtt{div}_1$ with $\mathtt{div}_2$. The reverse translation can usually replace $\mathtt{div}_2$ with $\mathtt{div}_1$ but not always. In translation-worthy data-expressions, it is typically sound; in formulas, it can easily be unsound because theorems about $\mathtt{div}_2$ might not require the restriction to non-zero denominators.

Alignment for Certain Arguments Two function symbols may be aligned only for certain arguments. This occurs if a_1 has a smaller domain than a_2.

The most fundamental case is the function type constructor $\to$ itself. For example, $\to_1$ may be first-order in F_1 and $\to_2$ higher-order in F_2. Thus, a translation from C_1 to C_2 can replace $\to_1$ with $\to_2$, whereas the reverse translation must be partial.

Another important class of examples is given by subtyping (or the lack thereof). For example, we could have

$$\begin{aligned} \mathtt{plus}_1 &: \mathtt{Nat} \to \mathtt{Nat} \to \mathtt{Nat} \\ \mathtt{plus}_2 &: \mathtt{Real} \to \mathtt{Real} \to \mathtt{Real} \end{aligned}.$$

Alignment up to Associativity An associative binary function (either logically associative or notationally right- or left-associative) can be defined as a flexary function, i.e., a function taking an arbitrarily long sequence of arguments. In this case, translations must fold or unfold the argument sequence. For example

$$\mathtt{plus}_1 : \mathtt{Nat} \to \mathtt{Nat} \to \mathtt{Nat} \qquad \mathtt{plus}_2 : \mathtt{List}\,\mathtt{Nat} \to \mathtt{Nat}.$$

All of the above types of alignments allow us to translate expressions between our corpora by modifying the lists of arguments the respective symbols are applied to, even if not always in a straight-forward way. The following types of alignments are more abstract, and any translation along them might be more dependent on the specifics of the symbols under consideration.

Contextual alignments Two symbols may be aligned only in certain contexts. For example, the complex numbers are represented as pairs of real numbers in some proof assistant libraries and as an inductive data type in others. Then only selected occurrences of pairs of real numbers can be aligned with the complex numbers.

Alignment with a Set of Declarations Here a single declaration in C_1 is aligned with a set of declarations in C_2. An example is a conjunction a_1 in C_1 of axioms aligned with a set of single axioms in C_2. More generally, the conjunction of a set of C_1-statements may be equivalent to the conjunction of a set of C_2-statements.

Here translations are much more involved and may require aggregation or projection operators.

Alignment between the Internal and External Perspective on Theories When reasoning about complex objects in a proof assistant (such as algebraic structures, or types with comparison) it is convenient to express them as theories that combine the actual type with operations on it or even properties of such operations. The different proof assistants often have incompatible mechanisms of expressing such theories including type classes, records and functors, with the additional distinction whether they are first-class objects or not.

We define the crucial difference for alignments here only by example. We speak of the internal perspective if we use a theory like

$$\texttt{theory}\,\texttt{Magma}_1 = \{u_1 : \texttt{Type},\ \circ_1 : u_1 \to u_1 \to u_1\}$$

and of the external perspective if we use operations like

$$\begin{aligned}&\texttt{Magma}_2 : \texttt{Type},\ u_2 : \texttt{Magma}_2 \to \texttt{Type},\\ &\circ_2 : (G : \texttt{Magma}) \to u_2\,G \to u_2\,G \to u_2\,G\end{aligned}$$

Here we have a non-trivial, systematic translation from C_1 to C_2. A reverse may also be possible, depending on the details of F_1.

Corpus-Foundation Alignment Orthogonal to all of the above, we have to consider alignments, where a symbol is primitive in one system but defined in another. More concretely, a_1 can be built-into F_1 whereas a_2 is defined in F_2. This is common for corpora based on significantly different foundations, as each foundation is likely to select different primitives. Therefore, it mostly occurs for the most basic concepts. For example, the boolean connectives, integers and strings are defined in some systems but primitive in others, as in some foundations they may not be easy to define.

The corpus-foundation alignments can be reduced to previously considered cases if we follow the "foundations-as-theories" approach [KR16], where the foundations themselves are represented in an appropriate logical framework. Then a_1 is simply an identifier in the corpus of foundations of the framework F_1.

Opaque Alignments The above alignments focused on logical corpora, partially because logical corpora allow for precise and mechanizable treatment of logical equivalence. Indeed, alignments from a logical into a computational or narrative corpus tend to be opaque: Whether and in what way the aligned symbols correspond to each other is not (or not easily) machine-understandable. For example, if a_2 refers to a function in a programming language library, that functions specification may be implicit or given only informally. Even worse, if a_2 is a wiki article, it may be subject to constant revision.

Nonetheless, such alignments are immensely useful in practice and should not be discarded. Therefore, we speak of opaque alignments if a_2 refers to a symbol whose semantics is unclear to machines.

Probabilistic Alignments Orthogonal to all of the above, the correctness of an alignment may be known only to some degree of certainty. In that case, we speak of probabilistic alignments. These occur in particular when machine-learning techniques are used to find large sets of alignments automatically. This is critical in practice to handle the existing large corpora.

The problem of probabilistically estimating the similarity of concepts in different corpora was studied before in [GK14]. We briefly restate the relevant aspects in our setting. Let T_i be the set of toplevel expressions occurring in C_i, e.g., the types of all constants and the formulas of all theorems. We assume a fixed set F of alignments, covering in particular the foundational concepts in F_1 and F_2.

Definition 1. *The pattern $P(f)$ of an expression f is obtained by normalizing f to $N(f)$ and abstracting over all occurrences of concepts that are not in F, resulting in $P(f) = \lambda c_1 \dots c_n.\ N(f)$. If two formulas $f \in T_1$ and $g \in T_2$ have α-equivalent patterns $\lambda d_1 \dots d_m.\ N(g)$ and $\lambda e_1 \dots e_m.\ N(h)$, we define their induced alignments by $I(f,g) = \{(d_1, e_1), \dots, (d_m, e_m)\}$. We write $J(p)$ for the union of all $I(f,g)$ with $P(f) =_\alpha P(g) =_\alpha p$.*

Example 1. For the formula $\forall x.\ x = 2 \cdot \pi \Rightarrow cos(x) = 0$ with F not covering the concepts 2, π, 0, and cos, and using a normal form N that exploits the commutativity of equality, we get the pattern $\lambda c_1\ c_2\ c_3\ c_4.\ \forall x.\ x = c_1 \cdot c_2 \Rightarrow c_3 = c_4(x)$.

Let $a_1, \dots, a_n$ be the set of all alignments in any $J(p)$. We first calculate an initial vector containing the similarities sim_i for each a_i by

$$sim_i = \sum_{\{p \ \mid \ a_i \in J(p)\}} \frac{1}{ln(2 + card\ \{\ f \mid P(f) = p\ \})}$$

Intuitively, an alignment has a high similarity value if it was produced by a large number of rare patterns.

Secondly, we iteratively transform this vector until its values stabilize. The idea behind this dynamical system is that the similarity score of an alignment should depend on the quality of its co-induced alignments. Each iteration step consists of two parts: we multiply the vector with the matrix

$$cor_{kl} = card\ \{\ (f,g) \mid\ \ a_k \in I(f,g) \land a_l \in I(f,g)\ \}$$

which measures the correlation between a_k and a_l, and then (in order to ensure convergence and squash all values into the interval $[0;1]$) apply the function $x \mapsto \frac{x}{x+1}$ to each component.

3 Global Identifiers

An essential requirement for relating logical corpora is standardizing the identifiers so that each identifier in the corpus can be uniquely referenced. It is desirable to use a uniform naming schema so that the syntax and semantics of identifiers can be understood and implemented as generically as possible. Therefore, we use MMT URIs [RK13], which have been specifically designed for that purpose.

3.1 General Structure

Syntax MMT URIs are triples of the form

`NAMESPACE ? MODULE ? SYMBOL`

The namespace part is a URI that serve as globally unique root identifiers of corpora, e.g. `http://mathhub.info/MyLogic/MyLibrary`. It is not necessary (although often useful) for namespaces to also be URLs, i.e., a reference to a physical location. But even if they are URLs, we do not specify what resource dereferencing should return. Note that because MMT URIs use ? as a separator, `MODULE ? SYMBOL` is the query part of the URI, which makes it easy to implement dereferencing in practice.[4]

The module and symbol parts of an MMT URI are logically meaningful names defined in the corpus: The module is the container (e.g., a signature, functor, theory, class, etc.) and the symbol is a name inside the module (of a type, constant, axiom, theorem etc.). Both module and symbol name may consist of multiple /-separated segments to allow for nested modules and qualified symbol names.

MMT URIs allow arbitrary Unicode characters. However, ? and /, which MMT URIs use as delimiters, as well as any character not legal in URIs must be escaped using the %-encoding. We refer to RFC 3986/7 for details.

3.2 Namespace Organization

MMT URIs standardize the syntax of the identifiers, but they still allow a lot of freedom how to assign URIs to the concepts in a specific corpus. This assignment is straightforward in principle — after all we only have to make sure that every concept has a unique URI. However, as we will see below, the structure of a corpus can pose some subtle issues that must be addressed carefully. Therefore, we quickly discuss commonly used corpus structures and how these can be used to form URIs systematically.

The common structuring feature of corpora is usually a directory tree. The leaves of this tree are files and contain modules. Moreover, each corpus usually has a certain root namespace. However, systems differ in how they subdivide a corpus into namespaces.

We distinguish the following cases:

[4] For simplicity in the remaining part of the paper we will not give complete HTTP links, but rather use single keyword abbreviations. Complete names of logics and modules are given in the online service.

- **flat** structure: All files share the same namespace regardless of their physical location in the directory tree. This naming schema is most well-known from SML. In this case, we can use the root namespace as the fixed namespace for all concepts in the corpus.
- **directory-based** structure: The namespace of a module is formed by concatenating the root namespace with the path to the directory containing it. There are two subcases regarding the treatment of the file name:
  - **files-as-modules**: Each file contains exactly one module, whose name is that of the file without the file name extension. The name of the module may be repeated *explicit*ly in the file or may be left *implicit*. Files as explicitly named modules is most well-known as the convention of Java.
  - **irrelevant file names**: The file name is irrelevant, i.e., the grouping of modules into files within the same directory is arbitrary. In particular, a file can contain multiple modules.
- **file-based** structure: The namespace of a module is formed by concatenating the root namespace of the corpus with the path to the file containing it.

3.3 URIs for Selected Proof Assistants

Using the principles defined above, we have developed MMT URI formation principles for some important proof assistants: **PVS** [ORS92], **Coq** [Coq15], and **Matita**, use directory-based namespaces, while **HOL4**, **Mizar** have one flat namespace. All of the systems define some kind of named, theory-like structure (e.g. articles in **Mizar**), which can be used for the module components. If modules are nested, we get module multi-part identifiers which are segmented by slashes. All modules declare symbols, whose names can directly be used in the symbol parts of the MMT URIs. If a symbol name N is declared multiple times in the same module (due to overloading), we use two-level names of the form N/i where i numbers all declarations of N in that module (starting at 1).

The exception to this are the HOL systems, in particular **HOL Light**, which does not have an obvious MMT URI formation principle because it does not maintain all its identifiers itself — instead it relies on the OCaml toplevel to store the assigned values. We use directory-based namespaces with files-as-modules. For constants and types introduced by a module we add the prefixes `const/` and `type/` respectively. If a file contains OCaml modules, we use their names to form multi-segment module names. Accordingly, if symbols result from OCaml structures, we form multi-segment symbol names. This has the effect that HOL Light URIs are formed in exactly the same way as for Coq. The other HOL systems can be treated similarly.

For informal collections of mathematical knowledge like Wikipedia, we can usually adopt similar measures. We interpret Wikipedia as having a flat namespace `http://en.wikipedia.org/wiki`, obtain modules via the files-as-models regime, and use the anchors of editable fragments as symbol names. Thus the statement of uniqueness of identity element and inverses in the Wikipedia article on groups would have the MMT URI `https://en.wikipedia.org/wiki?Group_(mathematics)?Uniqueness_of_identity_element_and_inverses`.

Details of the various definitions can be found in [MGK+17,PRA].

4 A Standard and Database for Alignments

Based on the observations of the previous sections, we now define a standard for alignments. Because many of the alignment types described in Section 2 are very difficult to handle rigorously and additional alignment types may be discovered in the future, we opt for a very simple and flexible definition.

Concretely, we use the following formal grammar for collections of alignments:

Collection	::=	(Comment | NSDef | Alignment)*
Comment	::=	// String
NSDef	::=	`namespace` String URI
Alignment	::=	URI URI (String = "String")*

Our definition aims at practicality, especially considering the typical case where researchers exchange and manipulate large collections of alignments. Therefore, our grammar allows for comments and for the introduction of short namespace definitions that abbreviate long namespaces. Our grammar represents each individual alignment as a pair of two URIs with arbitrary additional data stored as a list of key-value pairs.

The additional data in alignments makes our standard extensible: any user can standardize individual keys in order to define specific types of alignments. For example, for alignments up to argument order, we can add a key for giving the argument order. Moreover, this can be used to annotate metadata such as provenance or system versions.

In the sequel, we standardize some individual keys and use them to implement the most important alignment types from Section 2. In all definitions below, we assume that a_1 and a_2 are the aligned symbols.

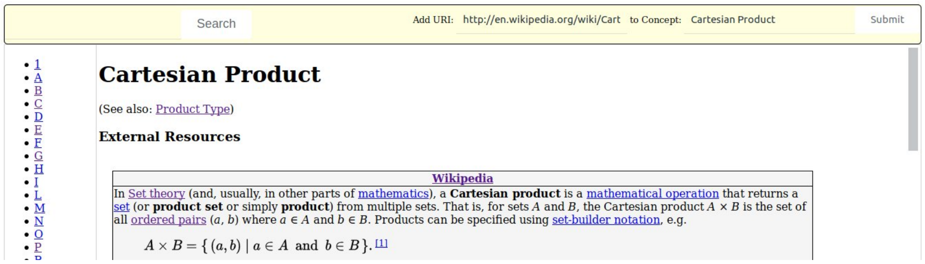
Definition 2. *The key* `direction` *has the possible values* `forward`*,* `backward`*, and* `both`*. Its presence legitimizes, respectively, the translation that replaces every occurrence of a_1 with a_2, its inverse, or both.*

Alignments with `direction` key subsume the alignment types of perfect alignments (where the direction is `both`) and the unidirectional types of alignment up to totality of functions or up to associativity, and alignment for certain arguments. The absence of this key indicates those alignment types where no symbol-to-symbol translation is possible, in particular opaque alignments.

Definition 3. *The key* `arguments` *has values of the form $(r_1, s_1) \ldots (r_k, s_k)$ where the r_i and s_i are natural numbers. Its presence legitimizes the translation of $a_1(x_1, \ldots, x_m)$ to $a_2(y_1, \ldots, y_n)$ where each y_k is defined by*

- *if $k = s_i$ for some i: the recursive translation of x_{r_i}*
- *otherwise: inferred from the context.*

Alignments with `arguments` key subsume the alignment types of alignments up to argument order and of alignment up to determined arguments.



Search
Add URI: http://en.wikipedia.org/wiki/Cart
to Concept: Cartesian Product
Submit
1
A
B
C
D
E
F
G
H
I
L
M
N
O
P

Cartesian Product

(See also: Product Type)

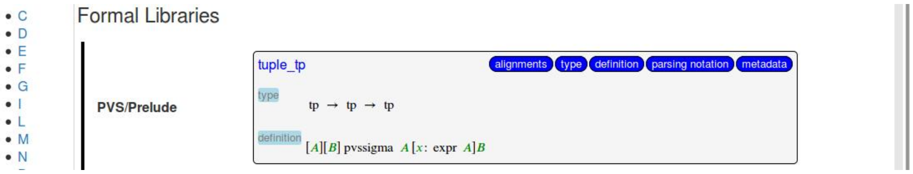
External Resources

Wikipedia

In Set theory (and, usually, in other parts of mathematics), a **Cartesian product** is a mathematical operation that returns a set (or **product set** or simply **product**) from multiple sets. That is, for sets A and B, the Cartesian product $A \times B$ is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. Products can be specified using set-builder notation, e.g.

$$A \times B = \{\,(a,b) \mid a \in A \ \text{ and } \ b \in B\,\}.^{[1]}$$

Fig. 1. The Alignment-based Dictionary — External Resources



Formal Libraries

PVS/Prelude

tuple_tp

type $tp \to tp \to tp$

definition $[A][B]$ pvssigma $A\,[x\!:\ \text{expr}\ \ A]B$

Fig. 2. The Alignment-based Dictionary — Formal Resources

Example 2. We obtain the following argument alignments for some of the examples from Section 2:

$$\begin{aligned}\texttt{Nat}_1\ \texttt{Nat}_2\ \texttt{direction} &= \text{"both"}\\ \texttt{eq}_1\ \texttt{eq}_2\ \texttt{arguments} &= \text{"}(1,2)(2,3)\text{"}\\ \texttt{contains}_1\ \texttt{in}_2\ \texttt{arguments} &= \text{"}(1,1)(2,3)(3,2)\text{"}\end{aligned}$$

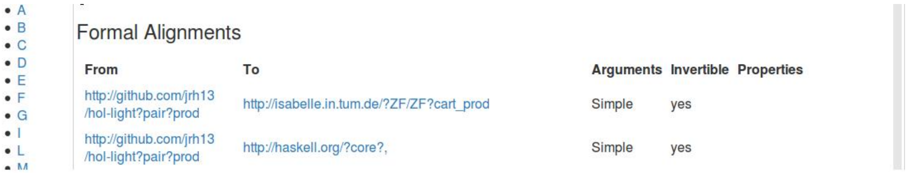
Finally, we standardize a key for probabilistic alignments:

Definition 4. *The key* `similarity` *has values that are real numbers in* $[0;1]$*. If used together with other keys like* `direction` *and* `arguments`*, it represents a certainty score for the correctness of the corresponding translation. If absent, its value can be assumed to be* 1 *indicating perfect certainty.*

We have implemented alignments in the MMT system [Rab13]. Moreover, we have created a public repository [PRA] and seeded it with a number of alignments (currently ≈ 12000) including the ones mentioned in this paper, the README of this repository furthermore describes the syntax for alignments above as well as the URI schemata for several proof assistants. The MMT system can be used to parse and serve all these alignments, implement the transitive closure, and (if possible) translate expressions according to alignments. Available alignments are shown in the MMT browser.

As an example service, we have started building an alignment-based math dictionary collecting formal and informal resources.[5] For this we extend the above grammar by the following:

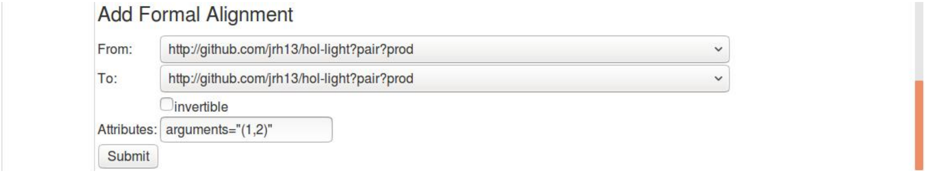
[5] https://mathhub.info/mh/mmt/:concepts?page=About



Formal Alignments

From	To	Arguments	Invertible	Properties
http://github.com/jrh13/hol-light?pair?prod	http://isabelle.in.tum.de/?ZF/ZF?cart_prod	Simple	yes	
http://github.com/jrh13/hol-light?pair?prod	http://haskell.org/?core?,	Simple	yes	

Fig. 3. The Alignment-based Dictionary — Available Alignments



Add Formal Alignment

From: http://github.com/jrh13/hol-light?pair?prod

To: http://github.com/jrh13/hol-light?pair?prod

□ invertible

Attributes: arguments="(1,2)"

Fig. 4. The Alignment based Dictionary - Field for Adding Alignments

$$\text{Alignment} \quad ::= \quad \text{String URI (String} = \text{"String"})^*$$

This assigns a mathematical concept (identified by the string) to a formal or informal resource (identified by the URI). The dictionary uses the above public repository, so additions to the latter will be added to the former. We have imported the ≈ 50,000 conceptual alignments from [GC14], although we chose not to add them to the dictionary yet, since the majority of them are (due to the different intention behind the conceptual mappings in Nnexus) dubious, highly contextual or otherwise undesirable.

Each entry in the dictionary shows snippets from select online resources if available (Figure 1), lists the associated formal statements (Figure 2) and available alignments between them (Figure 3), and allows for conveniently adding new individual URIs to concept entries as well as new formal alignments (Figures 1 and 4 respectively).

5 Conclusion

We have motivated and proposed a standard for aligning mathematical corpora. We presented examples of alignments between logical, computational, and semi-formal corpora and classified the different examples. The presented MMT-based system for sharing such alignments has been preloaded with thousands of alignments between the various kinds of concepts, including proof assistant types and constants, programming language (including computer algebra) algorithms, and semi-formal descriptions.

Future work includes extending the automated discovery of alignments [GK14] to foundations other than HOL. Our main focus was on the logical corpora, but we expect to be able to find much more opaque alignments. We invite the community to use the service. Finally we plan to integrate the use of the alignments database in the various mathematical knowledge management systems. In particular, we want to relate our methods and the alignment database to the tool chain for ontology alignment, e.g. the Alignment API [DESTdS11] or the work on logic-independent formalization of alignments in DOL [CMK14].

Acknowledgements We were supported by the German Science Foundation (DFG) under grants KO 2428/13-1 and RA-1872/3-1, the Austrian Science Fund (FWF) grant P26201, and the ERC starting grant no. 714034 *SMART*.

References

ACTZ06. A. Asperti, C. S. Coen, E. Tassi, and S. Zacchiroli. Crafting a Proof Assistant. In T. Altenkirch and C. McBride, editors, *TYPES*, pages 18–32. Springer, 2006.

AGC$^+$04. A. Asperti, F. Guidi, C. S. Coen, E. Tassi, and S. Zacchiroli. A content based mathematical search engine: Whelp. In J. Filliâtre, C. Paulin-Mohring, and B. Werner, editors, *Types for Proofs and Programs, International Workshop, TYPES 2004*, volume 3839 of *LNCS*, pages 17–32. Springer, 2004.

BFMP11. F. Bobot, J. Filliâtre, C. Marché, and A. Paskevich. Why3: Shepherd Your Herd of Provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*, pages 53–64, 2011.

CHK$^+$11. M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, and F. Rabe. Project Abstract: Logic Atlas and Integrator (LATIN). In J. Davenport, W. Farmer, F. Rabe, and J. Urban, editors, *Intelligent Computer Mathematics*, pages 289–291. Springer, 2011.

CMK14. M. Codescu, T. Mossakowski, and O. Kutz. A categorical approach to ontology alignment. In *Proceedings of the 9th International Conference on Ontology Matching*, pages 1–12. CEUR-WS.org, 2014.

Coq15. Coq Development Team. The Coq Proof Assistant: Reference Manual. Technical report, INRIA, 2015.

DESTdS11. J. David, J. Euzenat, F. Scharffe, and C. Trojahn dos Santos. The alignment api 4.0. *Semantic Web*, 2(1):3–10, 2011.

ESC07. J. Euzenat, P. Shvaiko, and E. Corporation. *Ontology matching*. Springer, 2007.

GC14. D. Ginev and J. Corneli. Nnexus reloaded. In Watt et al. [WDS$^+$14], pages 423–426.

GK14. T. Gauthier and C. Kaliszyk. Matching concepts across HOL libraries. In S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, editors, *CICM*, volume 8543 of *LNCS*, pages 267–281. Springer Verlag, 2014.

GK15. T. Gauthier and C. Kaliszyk. Sharing HOL4 and HOL Light proof knowledge. In M. Davis, A. Fehnker, A. McIver, and A. Voronkov, editors, *LPAR*, volume 9450 of *LNCS*, pages 372–386. Springer, 2015.

GKU16. T. Gauthier, C. Kaliszyk, and J. Urban. Initial experiments with statistical conjecturing over large formal corpora. In A. Kohlhase et al., editor, *Work in Progress at CICM 2016*, volume 1785 of *CEUR*, pages 219–228. CEUR-WS.org, 2016.

H$^+$15. T. C. Hales et al. A formal proof of the Kepler conjecture. *CoRR*, abs/1501.02155, 2015.

Hur09. J. Hurd. OpenTheory: Package Management for Higher Order Logic Theories. In G. D. Reis and L. Théry, editors, *Programming Languages for Mechanized Mathematics Systems*, pages 31–37. ACM, 2009.

KK13. C. Kaliszyk and A. Krauss. Scalable LCF-style proof translation. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP*, volume 7998 of *LNCS*, pages 51–66. Springer Verlag, 2013.

KR14. C. Kaliszyk and F. Rabe. Towards knowledge management for HOL Light. In Watt et al. [WDS+14], pages 357–372.

KR16. M. Kohlhase and F. Rabe. QED Reloaded: Towards a Pluralistic Formal Library of Mathematical Knowledge. *Journal of Formalized Reasoning*, 9(1):201–234, 2016.

KS10. A. Krauss and A. Schropp. A Mechanized Translation from Higher-Order Logic to Set Theory. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving*, pages 323–338. Springer, 2010.

KU15. C. Kaliszyk and J. Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, 2015.

KW10. C. Keller and B. Werner. Importing HOL Light into Coq. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving*, pages 307–322. Springer, 2010.

MGK+17. D. Müller, T. Gauthier, C. Kaliszyk, M. Kohlhase, and F. Rabe. Classification of alignments between concepts of formal mathematical systems. Technical report, 2017.

NSM01. P. Naumov, M. Stehr, and J. Meseguer. The HOL/NuPRL proof translator - a practical approach to formal interoperability. In R. Boulton and P. Jackson, editors, *14th International Conference on Theorem Proving in Higher Order Logics*. Springer, 2001.

ORS92. S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Springer, 1992.

OS06. S. Obua and S. Skalberg. Importing HOL into Isabelle/HOL. In N. Shankar and U. Furbach, editors, *Automated Reasoning*, volume 4130. Springer, 2006.

PRA. Public repository for alignments. https://gl.mathhub.info/alignments/Public.

Rab13. F. Rabe. The MMT API: A Generic MKM System. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *Intelligent Computer Mathematics*, pages 339–343. Springer, 2013.

RK13. F. Rabe and M. Kohlhase. A Scalable Module System. *Information and Computation*, 230(1):1–54, 2013.

WDS+14. S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, editors. *Intelligent Computer Mathematics*, number 8543 in LNCS. Springer, 2014.

Wie06. F. Wiedijk, editor. *The Seventeen Provers of the World*, volume 3600 of *LNCS*. Springer, 2006.

WPN08. M. Wenzel, L. C. Paulson, and T. Nipkow. The Isabelle framework. In A. Mohamed, Munoz, and Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2008)*, number 5170 in LNCS, pages 33–38. Springer, 2008.