A Standard for Aligning Mathematical Concepts

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

[1] University of Innsbruck
[2] 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, 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 would create a huge potential for knowledge sharing and transfer, e.g., integrating theorem provers or reusing services across systems.

Formally describing and verifying alignments is extremely expensive, as it typically requires not only aligning two concepts but whole libraries together with their foundations, or may even be impossible in case of alignments between semi-formal or informal concepts. Therefore, we introduce a lightweight approach that focuses on identifying the alignments while abstracting from formal definitions. Notably, this is already sufficient for many practically valuable applications.

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 to collect and curate alignments from the different kinds of mathematical corpora, including proof assistant libraries, computer algebra and programming language algorithms, and semi-formal libraries.

1 Introduction

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.

For ontologies, this problem has been relatively well-studied under the heading *ontology matching* [ESC07]. The situation in mathematics is somewhat special because mathematical knowledge involves rigorous notations, definitions, and properties, and attempts to capture it fully lead to very diverse corpora. Logical corpora have been developed in proof assistants and feature machine-understandable theorems and proofs. Computational corpora have been developed in computer algebra systems and feature executable definitions and constructions. And narrative corpora have been developed in wikis and related tools

featuring human-oriented semi-formal descriptions. For each kind, there are multiple large corpora, often the result of dozens of person-years of investment.

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.

Wiedijk [Wie06] explored a single theorem (and its proof) across 17 proof assistants implicitly providing alignments between the concepts present in the theorem's statement and proof. However, finding alignments has proved extremely difficult in general. There are three reasons for this: the conceptual differences between the three kinds of corpora; the differences between the underlying formal languages and tools; and the differences between the organization of the knowledge in the corpora.

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 second and fourth 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.

Due to the size of the involved corpora, it is desirable to find alignments automatically. Recently, the first author has developed heuristic methods for automatically finding alignments [GK14] targeted at integrating logical corpora [KK13] including HOL Light, HOL4, and Isabelle/HOL discovering 398 pairs of isomorphic concepts. Consistent name hashing combined with statement normalization was used to discover 39 symbols with equivalent definitions [KU15] in the Flyspeck development [H+15]. Ginev built a library of about 50,000 alignments between narrative corpora including Wikipedia, Wolfram Mathworld, PlanetMath and SMGloM [GC14]. Many practical services are enabled by even imperfect alignments, such as searching for a single query expression in multiple corpora at once, or providing more precise recommendations for automated reasoning [GK15].

Contribution and Overview Our contribution is two-fold. First, we present a phenomenological study of alignments between mathematical corpora in Section 2. Most importantly, we collect the various subtle reasons why an alignment may be imperfect because not all properties of the aligned symbols transfer exactly.

Our standardization includes a standardization of forming MMT URIs [RK13] for a number of a major logical corpora in Section 3. These assign a canonical URI to every symbol in a way that is unique across corpora and across logics. We use these URIs to give several examples from logical corpora in Section 4. The examples focus on logical corpora, but our results carry over to other kinds of corpora as well.

Second, we propose a standard for storing and sharing alignments in Section 5. Most corpora are developed and maintained by separate, often disjoint communities. That makes it difficult for researchers to utilize alignments because no central repository exists for jointly building a large collection of alignments. We have started such a central repository — 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. We describe this infrastructure in Section **??**.

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.

Perfect Alignment If a_1 and a_2 are logically equivalent modulo a translation φ 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:

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

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 $\texttt{Type}_1$ and $\texttt{Type}_2$, 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 only want certain formal properties to carry over, namely those in the *interface theory* in the sense of [KR16]. For example, in Section 4 we give many perfect alignments between symbols that use different but interface-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

$$\texttt{contains}_1 : (T : \texttt{Type}) \to \texttt{SubSet}\, T \to T \to \texttt{bool}$$

$$\texttt{in}_2 : (T : \texttt{Type}) \to \texttt{T} \to \texttt{SubSet}\, T \to \texttt{bool}$$

Here the expressions $\texttt{contains}_1(T, A, x)$ and $\texttt{in}_2(T, x, A)$ are aligned.

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

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

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.

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

$$\texttt{div}_1 : \texttt{Real} \to \texttt{Real} \to \texttt{Real} \qquad \texttt{div}_2 : \texttt{Real} \to (d : \texttt{Real}) \to \vdash d \neq 0 \to \texttt{Real}$$

Here inferring the third argument is undecidable, 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. $\texttt{div}_1$ might both have the type $\texttt{Real} \to \texttt{Real} \to \texttt{Real}$ with $x\, \texttt{div}_1\, 0$ undefined and $x\, \texttt{div}_2\, 0 = 0$.

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

$$\texttt{plus}_1 : \texttt{Nat} \to \texttt{Nat} \to \texttt{Nat} \qquad \texttt{plus}_2 : \texttt{Real} \to \texttt{Real} \to \texttt{Real}.$$

Another way for a_1 to have a smaller domain is to take less arguments. For example, we might have

$$\texttt{ln}_1 : \texttt{Real} \to \texttt{Real} \qquad \texttt{log}_2 : \texttt{Real} \to \texttt{Real} \to \texttt{Real}$$

where $\texttt{ln}_2(x)$ can be translated to $\texttt{log}_2(e, x)$.

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

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

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 Logical theories can be represented in two ways. We define them 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

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

Here we have non-trivial, systematical 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.

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/MyLibray`). 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.

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.

Both the corpus itself and the system with which it was processed may be subject to change. Therefore, it may be useful to record a version in an MMT URI. However, most developers take care to avoid semantically critical changes at least to the widely used parts of their corpora. Since those parts are also the most interesting ones for integration, we omit issues of versioning here and simply remark that the version can be recorded as a part of the root namespace.

Formation Principles The precise formation of MMT URIs may depend subtly on the foundational logic underlying the corpus. In the sequel, we identify some general principles that allow stating the formation rules concisely.

The most important physical structure of a corpus is usually a directory tree, whose leaves are files containing modules. In this case, the following principles are typical options to define namespaces:

- **flat** structure: All modules use the same namespace as the root namespace of the corpus regardless of their physical location in the corpus. This naming schema is most well-known from SML.
- **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**: The file contains exactly one module. The name of the module may be given *explicit*ly in the file or may be *implicit*. Either way, the name of the module must be the same as the file name without the file name extension. 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.

3.2 URIs for Selected Proof Assistants

Using the principles defined above, we describe the MMT URI formation principles for some important proof assistants. In all cases, we also assign MMT URIs for the underlying foundations in order to refer to built-in concepts.

PVS [ORS92] uses directory-based namespaces with irrelevant file names. We propose the following root URIs for some important PVS-related corpora:
- the PVS foundation: `http://pvs.csl.sri.com/foundation`
- the standard library that is shipped with PVS: `http://pvs.csl.sri.com/Prelude`
- the NASA corpus: `http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library`

Within a PVS corpus, the top-level modules are theories and (co)datatype declarations. The only possible nesting between them is that theories may contain (co)datatype declarations. Consequently, the module names have at most two segments. If a module is a (co)datatype, its symbols are the constructors, testers, etc. These are not hierarchical. If a module is a theory, its symbols are all declarations declared in it. These are not hierarchical. However, 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 from 1).

Coq [Coq15] uses directory-based namespaces with files as implicitly named modules. We propose the following root URIs:
- the Coq foundation: `https://coq.inria.fr/foundation`
- the standard library shipped with Coq: `https://coq.inria.fr/theories`
- the Coq contributions: `https://coq.inria.fr/contribs`
- the Mathematical Components corpus (including SSReflect): `http://ssr.msr-inria.inria.fr/math-comp`

Coq modules can be nested. Besides the module name given implicitly by the file, Coq files can contain modules and module types. Symbols are all declarations inside a module. Their names can be hierarchic due to generative functor instantiation.

Matita uses the same URIs as Coq except for not allowing nested modules. We suggest the following root URIs:
- the Matita foundation: `http://matita.cs.unibo.it/foundation`
- the Matita standard library: `http://matita.cs.unibo.it/library`

Mizar uses a flat namespace. We propose the following root URIs:
- the Mizar foundation: `http://mizar.org/foundation`
- the Mizar Mathematical Library (MML): `http://mizar.org/library`

The Mizar modules are the articles. The name of a module is the name of the article without the file name extension. There is no nesting of modules. The Mizar symbols are all the declarations inside an article. Their names are obtained through a heavily idiosyncratic naming schema that includes generating unique names by numbering the declarations of the same kind in each article. For example, the MMT URI of conjunction is `http://mizar.org/corpus?XBOOLEAN0?K4`

HOL Light 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. There are three kinds of HOL Light symbols: types, constants, and theorems; only the former two are visible to the HOL Light kernel. For each type or constant, we use the name visible to the HOL Light

kernel. For each theorem, we use the OCaml binding name. OCaml toplevel symbols may be grouped into OCaml modules. This feature is seldom used, as it only affects theorem names: the kernel is also not aware of the OCaml module in which the definitions of constants or types are introduced.

This has the effect that different HOL Light files can be incompatible with each other. There are two reasons for this incompatibility: First, toplevel symbols may be overwritten by others, which means that the original ones are no longer accessible and a formalization might fail. This happens for example in case of the OCaml basic output function `open_in` which gets overwritten by a theorem with the same name. Second, the loading of a file may change the state of HOL Light kernel or packages to one that is no longer compatible with another file [Wie09]. For example the theories `complexnumbers.ml` and `complexes.ml` both introduce the type `complex` but with different definitions.

Thus, HOL Light's names do not uniquely identify symbols. Therefore, 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.

We propose the following root URIs:

- the HOL Light foundation and the library shipped with it:
  `http://github.com/jrh13/hol-light`
- the Formal Proof of Kepler formalization [H+15]:
  `http://github.com/flyspeck/flyspeck`

For example, the Flyspeck theorem `well_defined_unordered_pair` is assigned the URI: `http://github.com/flyspeck/flyspeck?text_formalization/packing/marchal3?Matchal_cells_3.well_defined_unordered_pair`.

HOL4 internal module names correspond to names of files, however the names of types and constants are not associated with the modules. Furthermore, the names of constants and types are separate. To avoid ambiguity we use the same module names as for HOL Light. We propose the following root URIs:

- the HOL4 foundation and the library shipped with it:
  `https://hol-theorem-prover.org`

Isabelle is a logical framework: Its distribution includes a number of object logics, and each Isabelle theory uses an object logic (or declares a new one). Isabelle uses files as explicitly named modules. However, it disregards the directory structure: The system makes sure that two modules with same name cannot be loaded in the same session even if they are stored in different directories. But as different object logics and different developments often declare incompatible notions, we still use directory-based namespaces to make sure all theories have unique namespaces.

Isabelle allows several module mechanisms including locales and type classes [HW06]. Therefore, we form nested module names by concatenating theory and locale/type class names. We propose the following root URIs:

– Isabelle foundation and distributed libraries: http://isabelle.in.tum.de/
– the Archive of Formal Proofs: http://afp.sf.net/
– the TLA+ logic: http://tla.msr-inria.inria.fr/

For example the type of streams is represented by the URI:
http://isabelle.in.tum.de/?HOL/corpus/Stream?stream

4 Examples of Alignments

Using the MMT URIs defined in Sect. 3, we give a detailed presentation of alignments across proof assistants for three representative concepts. We also include some alignments to programming languages, which are relevant for code generation. In all cases, we will see how big the differences between the details are across the logical corpora even though most of the alignments are in fact perfect.

Cartesian Product In constructive type theory, there are two common ways of expressing the non-dependent Cartesian product. First, if the foundation has inductive types such as the Calculus of Inductive Constructions, it can be an inductive type with one binary constructor. Second, if the foundation has a dependent sum type, Cartesian products can be the non-dependent special case. The first two symbols below use the former, the last one the latter approach:

– http://coq.inria.fr/theories?Init/Datatypes?prod.ind
– http://matita.cs.unibo.it/?datatypes/constructors?Prod.ind
– http://isabelle.in.tum.de/?CTT/CTT?times

In higher-order logic, the only way to introduce types is by using the `typedef` construction, which constructs a new type that is isomorphic to a certain subtype of an existing type. In particular, most HOL systems introduce the Cartesian product $A \times B$ by using an appropriate unary predicate on $A \to B \to$ `bool`:

– http://github.com/jrh13/hol-light?pair/type?prod
– http://hol-theorem-prover.org/?pair/type?prod
– http://isabelle.in.tum.de/?HOL/Product?prod

In PVS, the product type constructor is part of the system itself:

– http://pvs.csl.sri.com/foundation?PVS?tuple_tp

In set theory, it is also possible to restrict dependent sum types to obtain the Cartesian product. This is used in Isabelle/ZF. In Mizar the Cartesian product is defined implicitly as a first order functor, which involves discharging the well-definedness condition. Therefore, we give the URIs of both the definition and the generated functor.

– http://isabelle.in.tum.de/?ZF/ZF?cart_prod
– http://mizar.org/library/?ZFMISC_1?K2
(defined by http://mizar.org/library/?ZFMISC_1?def_2)

Finally, Cartesian products appear in most programming languages and we list here a number of constructions that can be aligned:

– http://caml.inria.fr/?core?*
– http://haskell.org/?core?,

– http://scala-lang.org/?core?,
– http://cppreference.com/?std?pair

Informal sources that can be aligned are e.g.:

– https://en.wikipedia.org/wiki/Cartesian_product
– https://en.wikipedia.org/wiki/Product_type
– http://mathworld.wolfram.com/CartesianProduct.html

Addition In constructive type theory, addition of natural numbers is typically defined as a fixed points of certain equations.

– http://coq.inria.fr/theories?Init/Nat?add
– http://matita.cs.unibo.it/?nat/plus?plus

Higher-order logic proof assistants usually use high-level constructions for primitive recursion. Interestingly, here the alignment is very obvious at this high-level even though the elaboration into core language features may result in very different-looking, but logically equivalent definitions. This is how addition is implemented in HOL Light and HOL4. Isabelle/HOL uses a similar construction but inside a type class for commutative monoids with difference.

– http://github.com/jrh13/hol-light?arith/const?+
– http://hol-theorem-prover.org/?arithmetic/const?+
– http://isabelle.in.tum.de/?HOL/Nat?plus_nat_inst.plus_nat

In set theory, defining addition is surprisingly the least straightforward: In Isabelle/ZF, addition uses the `primrec` construction together with a coercion from non naturals to naturals. PVS defines a general addition on number fields, which all the concrete number spaces (reals, integers, etc.) inherit. In Mizar, it is the restriction of complex addition to natural numbers. Mizar's complex addition itself is built from the real number addition, which in turn is built from positive real addition, rational addition, and ordinal addition. So in principle it would be possible to align with Mizar's ordinal addition.

– http://pvs.csl.sri.com/Prelude?number_fields?+
– http://isabelle.in.tum.de/?ZF/Arith?add
– http://mizar.org/library/?NAT_1?K2

Most programming languages do not implement arbitrary precision natural numbers. However, it is possible to find partial mappings, which are in fact already used by efficient code generation [HN10] for Isabelle/HOL natural numbers and the representation of Coq natural numbers by its extended bytecode machine [AGST10].

– http://caml.inria.fr/?Big_Int?add_big_int
– http://haskell.org/?core?+
– http://www.smlnj.org/?IntInf?+

Concatenation of Lists In constructive type theory (e.g. for Matita, Coq), the append operation on lists can again be defined as a fixed point. In higher-order logic, append for polymorphic lists can be defined by primitive recursion, as done by HOL Light and HOL4. Isabelle/HOL slightly differs from these two because it uses lists that were built with the co-datatype package [B+14].

– http://coq.inria.fr/theories?Init/Datatypes?app

– http://github.com/jrh13/hol-light?lists/const?APPEND
– http://hol-theorem-prover.org/?list/const?APPEND
– http://isabelle.in.tum.de/?HOL/List?append

In set theory, PVS and Isabelle/ZF also use primitive recursion for monomorphic lists. In Mizar, lists are represented by finite sequences, which are functions from a finite subset of natural numbers (one-based `FINSEQ` and zero-based `XFINSEQ`) with append provided.

– http://pvs.csl.sri.com/Prelude?list_props?append
– http://isabelle.in.tum.de/?ZF/List_ZF?app
– http://mizar.org/library/?ORDINAL4/K1

Concatenation of lists is also common in programming languages.

– http://caml.inria.fr/?core?@ http://haskell.org/?core?++
– http://scala-lang.org/?core?++

5 A Standard and Databse for Alignments

Based on the observations of the previous sections, we now define a standard for alignments that covers the practically relevant examples. We use the following formal grammar for collections of alignments:

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

Here NSDef defines abbreviations for CURIEs (as defined by the W3C), which allows shortening URIs with the same long namespaces. An alignment is just a pair of URIs with a list of key-value pairs, which allows adding author/source, certainty scores, translation instructions, etc.

We also standardize some special keys and possible values that are important for practical applications. As a guiding criterion for defining these keys, we use how and in which directions expressions with head s_1 or s_2 can be translated.

The simplest case is the following:

Definition 1. *A **simple alignment** uses the key* `direction` *with the possible values* `forward`, `backward`, *and* `both`. *It induces the translation that replaces every occurrence of s_1 with s_2, or of s_2 with s_1 according to the value of the key.*

This subsumes perfect alignments (where the direction is `both`) and several unidirectional cases: alignment up to totality of functions or up to associativity, and alignment for certain arguments. The absence of this key indicates alignments where no translation is possible, in particular opaque alignments.

The following case covers alignments up to argument order or determined arguments:

Definition 2. *An **argument alignment** uses the key* `arguments` *whose value A is of the form $(i, j)^*$ where i and j are natural numbers.*

It induces the translation of $s_1(x_1, \ldots, x_m)$ to $s_2(y_1, \ldots, y_n)$ where y_j is the recursive translation of x_i if (i, j) is among the pairs in A and inferred from the context if there is no i for which (i, j) is in A.

Example 1. We obtain the following argument alignments for some of the examples from Section 2:

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

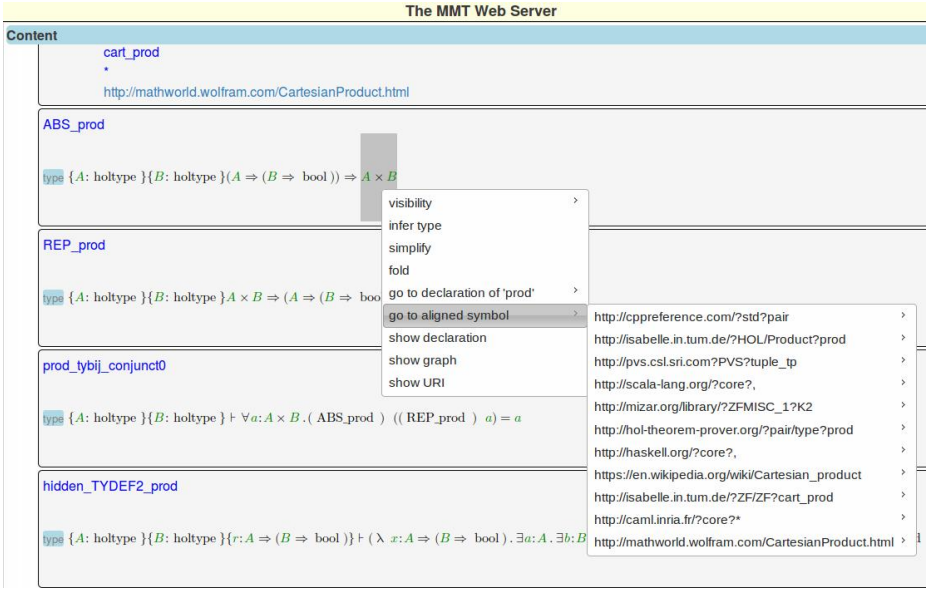

Fig. 1. The MMT Server Showing Formal and Informal Alignments

We have implemented alignments in the MMT system [Rab13]. Moreover, we have created a public repository[3] and seeded it with a number of alignments including the ones mentioned in this paper. The MMT system can be used to read and serve all these alignments, implement the transitive closure, and (if possible) translate expressions according to alignments.

As an example service, we have added alignment support to the MMT web browser: Figure 1 shows a screenshot from browsing the HOL Light library, in particular a snapshot of the `pairs` module. The symbol `prod` is aligned with several formal and informal sources, which can be shown and navigated to by right-clicking the symbol in an expression (here in the type of the symbol `ABS_prod`).

[3] https://gl.mathhub.info/alignments/Public

6 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.

Acknowledgements We acknowledge financial support from the German Science Foundation (DFG) under grants KO 2428/13-1 and RA-1872/3-1.

References

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

AGST10. Michaël Armand, Benjamin Grégoire, Arnaud Spiwack, and Laurent Théry. Extending Coq with imperative features and its application to SAT verification. In Matt Kaufmann and Lawrence C. Paulson, editors, *ITP*, volume 6172 of *LNCS*, pages 83–98. Springer, 2010.

B$^+$14. Jasmin Christian Blanchette et al. Truly modular (co)datatypes for Isabelle/HOL. In Gerwin Klein and Ruben Gamboa, editors, *ITP*, volume 8558 of *LNCS*, pages 93–110. Springer, 2014.

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.

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

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

GC14. Deyan Ginev and Joseph Corneli. Nnexus reloaded. In Stephan Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics 2014*, number 8543 in LNCS, pages 423–426. Springer, 2014.

GK14. Thibault Gauthier and Cezary Kaliszyk. Matching concepts across HOL libraries. In Stephen Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban, editors, *CICM*, volume 8543 of *LNCS*, pages 267–281. Springer Verlag, 2014.

GK15. Thibault Gauthier and Cezary Kaliszyk. Sharing HOL4 and HOL Light proof knowledge. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *LPAR*, volume 9450 of *LNCS*, pages 372–386. Springer, 2015.

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

HN10. Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In Matthias Blume, Naoki Kobayashi, and Germán Vidal, editors, *FLOPS*, volume 6009 of *LNCS*, pages 103–117. Springer, 2010.

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

HW06. Florian Haftmann and Makarius Wenzel. Constructive type classes in Isabelle. In Thorsten Altenkirch and Conor McBride, editors, *Types for Proofs and Programs, International Workshop, TYPES 2006*, volume 4502 of *LNCS*, pages 160–174. Springer, 2006.

KK13. Cezary Kaliszyk and Alexander Krauss. Scalable LCF-style proof translation. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *ITP*, volume 7998 of *LNCS*, pages 51–66. Springer Verlag, 2013.

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. Cezary Kaliszyk and Josef 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.

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.

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.

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

Wie09. Freek Wiedijk. Stateless HOL. In Tom Hirschowitz, editor, *TYPES*, volume 53 of *EPTCS*, pages 47–61, 2009.

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