Aligning Concepts across Proof Assistant Libraries

Thibault Gauthier

Department of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria

Cezary Kaliszyk

Department of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria

Abstract

As the knowledge available in the computer understandable proof corpora grows, recognizing repeating patterns becomes a necessary requirement in order to organize, synthesize, share, and transmit ideas. In this work, we automatically discover patterns in the libraries of interactive theorem provers and thus provide the basis for such applications for proof assistants. This involves detecting close properties, inducing the presence of matching concepts, as well as dynamically evaluating the quality of matches from the similarity of the environment of each concept. We further propose a classification process, which involves a disambiguation mechanism to decide which concepts actually represent the same mathematical ideas.

We evaluate the approach on the libraries of six proof assistants based on different logical foundations: HOL4, HOL Light, and Isabelle/HOL for higher-order logic, Coq and Matita for intuitionistic type theory, and the Mizar Mathematical Library for set theory. Comparing the structures available in these libraries our algorithm automatically discovers hundreds of isomorphic concepts and thousands of highly similar ones.

Key words: proof assistant libraries; library alignment; higher-order logic; type theory; set theory; dynamical systems

Email addresses: thibault.gauthier@uibk.ac.at (Thibault Gauthier), cezary.kaliszyk@uibk.ac.at (Cezary Kaliszyk).

URLs: http://cl-informatik.uibk.ac.at/users/tgauthier/ (Thibault Gauthier), http://cl-informatik.uibk.ac.at/cek/ (Cezary Kaliszyk).

1. Introduction

1.1. Context

With the diversity of interactive theorems provers (Harrison et al., 2014), the lack of interoperability is a growing issue. Formalized proofs originating from one prover are hardly reusable in a different one. Discovering and identifying the structures that occur in multiple libraries becomes an important step to better interoperability as the libraries of theorem provers grow.

The benefits of links between different structures have since long been known by mathematicians (Corry, 2012). Algebraic structures such as fields (Rotman, 2010) enable mathematicians to transport properties from real to complex numbers. Moreover the whole field of category theory has been about generalization (Awodey, 2006) with recent techniques such as classifying a topos of a theory as very powerful transfer mechanism (Univalent Foundations Program, 2013). In computer programming, oriented-object languages (Meyer, 1988) can share a method across many object instances using inheritance. Both examples shows how an interconnected structure is beneficial for better insights and faster development.

To this end, we develop an algorithm that automatically evaluates the similarity between formalized concepts (units of thought). This is achieved by inferring the mathematical properties they possess, which is a reflection of the structure they describe or belong to.

1.2. Challenges

Aligning libraries comes with a set of challenges. The mere fact that common mathematical structures have been (re-)formalized in each proof assistant makes this initiative conceivable.

The first difficulty is to express the mathematical properties uniformly. The multiplicity of the logics of the studied provers make this step quite complicated. Indeed, they have often different degree of support for lambda-abstractions, polymorphism, type classes, type hierarchies, algebraic hierarchies, etc. Those features produce some idiosyncratic constructions in the formal developments in each prover.

The next step is to define and recognize which mathematical concepts appear in the library. There may be for instance types, constants, subterms, formula subtrees or even proof tactics. Our goal will be to define what are the unit concepts and which ones are a combination of those concepts. Another issue is that some concepts are defined many times inside one library. Indeed different integer representations can be more suitable for some applications (like code extraction (Haftmann et al., 2013)). Conversely, a concept can belong to many different structures. It is especially common in the traditional set theoretic approach, where the empty set $\emptyset$ also stands for the natural number 0. This is realized by most formalizations of set theory, for example in the foundations of Mizar (Naumowicz and Korniłowicz, 2009) and Isabelle/ZF (Paulson, 2016).

Having delimited our notion of "concepts", we wish to derive their similarities. A uniform representation for the properties makes it easy to infer which concepts share the same properties. We would like to emphasize here that the approach is more effective and more comprehensive than looking only at their definitions. Already for minimally different definitions, recognizing that they represent the same concept is not straightforward.

This becomes very hard when definitions are foundationally different, for instance the real numbers may be defined through Dedekind cuts or Cauchy sequences. Moreover, the similarity measure may indicate for example the discovery of the underlying ring structures of integers and real numbers, which would not be possible if we restrict to the discovery of perfect matches only. Furthermore, the context in which the concept is expressed can be essential. To capture its influence, we also study the interconnections between properties inside a library that allow finding similar relations between concepts in different libraries.

We hope that solving these issues will create libraries of alignments suitable for the different types of applications envisioned.

1.3. Applications

The principal application of our work is transferring theorems between libraries. Deep embeddings (Jacquel et al., 2015) are typically best-suited to check the soundness of the provers and prove meta-theorems about the system studied. Yet, the imported theorems are difficult to integrate with the current developments since they are created at different logical levels. Therefore, shallow embeddings (Myreen and Davis, 2014) that can be obtained through reflection (Keller and Werner, 2010) are preferred. Even then, if no concept mapping is performed, the potential risk is to create parallel developments on the same set of concepts. And additionally to the unnecessary repetitions, the equivalence between the two sets would have to be proven, which may not be possible. From our discovered alignments, it is possible to lift the set of mappings found to theorems and proofs. This yields a possible translation that can be used to import a library into another system in a sensible manner by reusing the common concepts. Still, each translated theorem needs to be derived in the other prover and porting proofs is a difficult process due to the possible differences in definitions. In our previous work (Gauthier and Kaliszyk, 2015), we relied on the matching algorithm to transfer proof knowledge between two HOL systems and evaluated how it improved the success of a machine-learning based proving framework HOL(y)Hammer (Kaliszyk and Urban, 2014b).

The second set of applications arises from the fact that our alignment procedure could also be used effectively within a single library. A first practical use is the removal of duplicate constants and theorems. Another possibility is to hyper-link similar objects to create a better proving environment. Moreover, the properties shared by similar concepts can be combined into a structure and the concepts made instances of this structure. In the case of types, this could lead to a possible refactoring of the type hierarchy present in the system which could be essential to share proofs across different domains. Proof assistants attempt to maximize sharing. This idea is most visible in proof assistants based on type theory such as Lean (de Moura et al., 2015) where its automation relies on its library structure. But it has been at the basis of one of the earliest proof assistant Mizar (Bancerek et al., 2015). Using fuzzy mappings inside one library, initial experiments on the possibility of producing new conjectures from analogues of theorems of a related domain were performed in (Gauthier et al., 2016). Finally, various proof refactoring techniques (Whiteside et al., 2011; Dietrich et al., 2013; Klein, 2014) rely on similarities between concepts, such as these found here. Also refactoring may benefit from the patterns in the formalizations revealed by our algorithm.

1.4. Contributions

This paper is an extended version of our work presented at CICM 2014 (Gauthier and Kaliszyk, 2014) which introduced a simple concept matching algorithm for a single foundation (higher-order logic). In this paper we present many extensions of this work, which allow much better automatic discovery of isomorphic and similar structures in multiple formal mathematical libraries based on different foundations. Specifically the contributions of this work are:

- We design a fixpoint algorithm with various scoring functions for automatic discovery of similar concepts and theorems in and across proof assistant libraries. We find thousands of mappings between concepts. These include one-to-many mappings where concepts are related to multiple counterparts.
- We build properties and concepts from the objects of formal libraries: theorems, constants and subterms. During this step, we experiment with various degrees of normalization and an optional conceptualization of subterms, as well as different level of type inclusion.
- We evaluate the proposed approaches on the libraries originating from 6 interactive theorem provers based on different foundations including set theory and type theory. We translate them into a common representation, manually aligning the term representations of the different logics.
- We give an interpretation of the correlation between matches used in our fixpoint algorithm, and show that it can also be a key idea to produce sensible mappings for formulas.
- We investigate the possibility of using an intermediate library as a translation between two libraries by constructing transitive matches.
- We define various degree of subjective similarities given by the mappings. The highest degree is defined as an optimal matches. It happens when the two related objects represent the same object conceptually.
- We describe a classification algorithm that decides which matches are optimal. We produce hundreds of such optimal matches for each pair of libraries.

1.5. General principles of the algorithm

Our algorithm takes as an input the objects of two proof assistant libraries. These objects are types, constants and theorems. We do not consider the proofs. The aim of the algorithm is to recognize the constants (including types), or more generally subterms, representing the same (and/or related) concept occurring in different libraries. We will use properties such as associativity or nilpotence, extracted from the term representation of theorems to evaluate the similarity of two constants. A general guideline is that the more properties two constants have in common the more similar they are. In addition to this main idea, relations between similarity pairs together with a number of heuristics help refine the accuracy of our similarity measures. This means, that we will use a self-improving mechanism, called dynamical scoring, where each similarity pair is influenced by the similarities of other pairs. For instance, the strength of a matching between $<$ and $\subset$ is correlated with the degree of similarities of the constants 0 and $\emptyset$. The result is a list of pair of constants, sorted by their similarity scores. On top of that, a procedure can be applied to decide if the best scoring match should be in the final mappings. This procedure relies on additional techniques such as disambiguation or type coherence. If

the choice is not delayed, the score of the match is raised or diminished according to the decision. This in turn influences further applications of the dynamical scoring algorithm. An overview of the proposed procedure is presented in Fig. 1.

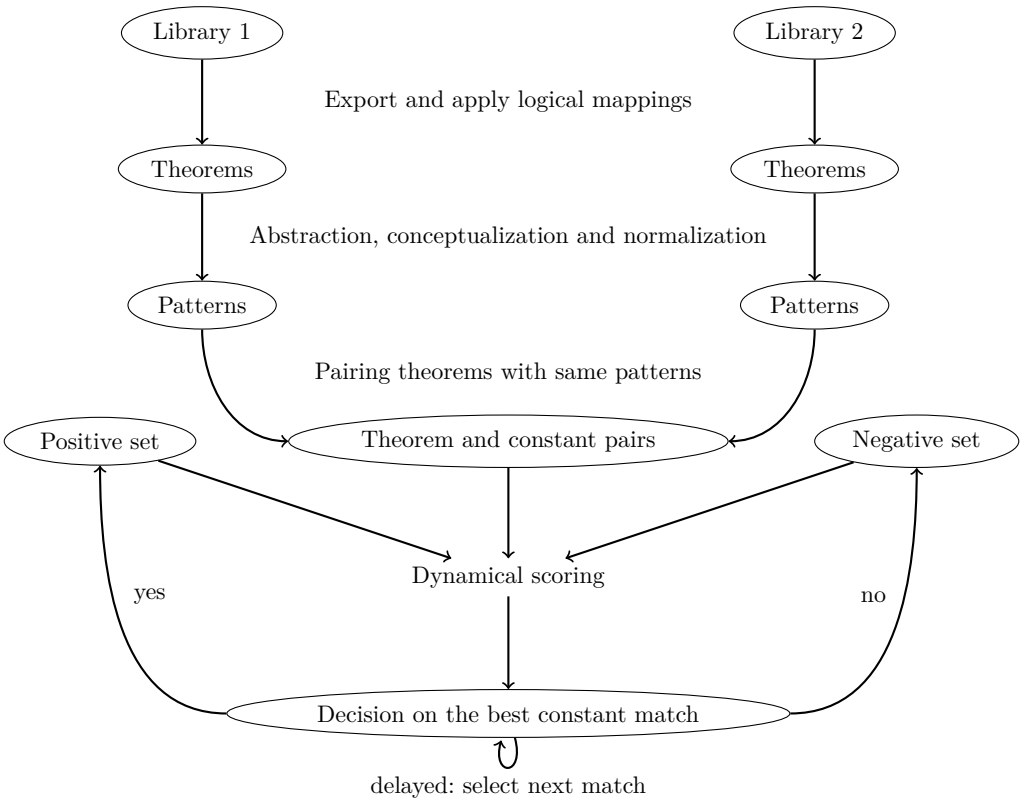

Fig. 1. Workflow graph

1.6. Plan

The rest of this paper is organized as follows. In Section 2 we explain the process of creating concepts and properties inside one prover. We describe how we match properties and deduce similarity scores between concepts in Section 3. In Section 4 we evaluate each step of our approach. We next describe additional techniques applied on top of the scoring procedure that improve the quality of our results (Section 5). We discuss related work in Section 6. In Sections 7 and 8 we conclude and present an outlook on the future work.

2. Creating properties and concepts from theorems

The only prerequisite of our algorithms, is a common term representation of theorems in the considered proof assistant libraries. This requirement is immediately satisfied when considering matching of concepts in different formalizations or proof libraries of one system, but it may be harder to satisfy for libraries or proof assistants based on different

foundations. We focus on the term structures, or the syntactic structures rather than the the semantics of the formulas in order to work across the different logical foundations. The term structures are exported from the internal term representation in each prover and thus contains implicit arguments and coercions that are not visible in the external syntax. This additional information makes detecting alignments more challenging but produces more precise mappings.

We believe that aligning concepts using this approach rather than providing a deep embedding is more appropriate. Indeed, the proof assistants have been meant to help proof developers and therefore their syntax is usually designed to correspond to standard mathematics. This also implies that the libraries created by the users often state theorems in a similar way. Moreover, each formal proof library is completely self-sufficient which means that basic types, such as integers, real numbers, or sets, are likely to be defined in all the proof libraries, which asserts that certain concept alignments do exist. Since the logical operators are often tied with intricacies of the logic, they usually need to be recognized and mapped manually (see Section 4.1). In all our experiments we will assume that the representation of the terms corresponds to a version of type theory that includes the basic predicate logic and equality. Therefore, we manually recognize the constants $\forall, \exists, \Rightarrow, \neg, \wedge, \vee, \Leftrightarrow, =$. We additionally map the the type of propositions, which we will denote as $\$o$ and the type of all types $\$t$. The names $\$o$ and $\$t$ are the TPTP (Sutcliffe et al., 2012) notations used for the two types. These mappings actually collapse the type hierarchy. This however has no consequence for the algorithm: the intent is to discover concept similarities, and only proving or disproving their equality requires a sound system. Furthermore, even if the found concept pairs are not equivalent, but only similar, such pairs are still often useful.

With a common representation of theorems, we can identify the theorems that are instances of the same property. Since two statements may represent the same property even if they are presented in different forms, we normalize the statements of the theorems. Furthermore, the found properties should not depend on the name of the constants present in the statement, therefore constants should be abstracted after normalization. From this intuition, we now give a formal definition of a property.

Definition 1 (Property)**.** Given a set of terms $\mathcal{T}$, and a normalization method $N : \mathcal{T} \to \mathcal{T}$, a property P of a theorem $T \in \mathcal{T}$ is defined by:

$$P =_{def} \lambda C_1, \ldots, C_n.\ N(T)$$

where $C_1, \ldots, C_n$ are the non-logical constants appearing in $N(T)$ ordered by a left outermost traversal of $N(T)$. Two properties will be said equal if they are α-equivalent.

Definition 2 (Derived matchings)**.** Two theorems T_1 and T_2 which share the same property P are called a matching pair of theorems. Let (T_1, T_2) be a a matching pair of theorems with normalized forms $N(T_1) = P(D_1, \ldots, D_n)$ and $N(T_2) = P(E_1, \ldots, E_n)$. The matching pairs of constants $(D_1, E_1), \ldots, (D_n, E_n)$ are induced by the pair (T_1, T_2). We will also say that two constants D and E have the same property if they occur at the same position in two equal properties. The similarity of D and E will be measured by the number and quality of these properties.

Remark. The distinction made in our previous work (Gauthier and Kaliszyk, 2014) between patterns of theorems and properties of constants is now subsumed by this single definition. Patterns of theorems are now also called properties. The distinction between different positions inside a property is now defined implicitly by the process of inducing pairs of constants. These changes lead to a much more concise description and a significant gain in memory and speed for our algorithms.

Example. Given the below theorems T_1 and T_2, their respective normalizations, and the properties extracted from their statements:

$$T_1 : \ \forall x : \mathit{num}.\ x + 0 = x \qquad T_2 : \ \forall x : \mathit{real}.\ x = x \times 1$$
$$N(T_1) : \ \forall x : \mathit{num}.\ x = x + 0 \qquad N(T_2) : \ \forall x : \mathit{real}.\ x = x \times 1$$
$$P_1 : \lambda \mathit{num}, +, 0.\ \forall x : \mathit{num}\ x = x + 0 \qquad P_2 : \lambda \mathit{real}, \times, 1.\ \forall x : \mathit{real}.\ x = x \times 1$$

The properties P_1 and P_2 are α-equivalent, therefore the theorems T_1 and T_2 form a matching pair of theorems, and the following three matching pairs of constants are derived:

$$\mathit{num} \leftrightarrow \mathit{real},\ + \leftrightarrow \times,\ 0 \leftrightarrow 1$$

The purpose of a normalization method is to maximize the number of shared properties without sacrificing the characteristics of each individual one. This is typically done by rewriting the theorems into a normal form and by extending the types of the considered concepts. We will first focus on the rewriting based methods: computing conjunctive normal forms, reordering commutative and associative-commutative connectives, and normalizing subterms. Then we will discuss different levels of typing that can be applied and how they interact with the normalization methods.

The effect of the rewriting based methods is illustrated on a running example. For clarity, type information is omitted, constants are not abstracted.

Example. (Running) The constants $\times$, *alt_pi*, s, 0, *cos* and fst respectively stand for multiplication, π, successor, zero, cosine and projection on the first argument.

$$\forall y\ x.\ x = \mathit{alt\_pi}\ \times (s\ (s\ 0)) \Rightarrow \mathit{cos}\ x = \mathrm{fst}\ 0\ y$$

2.1. Conjunctive normal forms

First, we split the theorems into separate conjuncts even when they appear under quantifiers. Each conjunct can be considered as separate theorems from this point. We then rewrite every theorem statement to conjunctive normal forms. In this normalization we assume classical logic, however if this is not desired, it is possible to consider intuitionistic clausification (Otten, 2005). As the focus is on proof libraries that can be expressed in type theory, we preserve the equivalences $\Leftrightarrow$ and consider them as equalities between propositions.

Example. (Running) $\forall y\ x.\ \neg(x = \mathit{alt\_pi} \times (s\ (s\ 0))) \vee \mathit{cos}\ x = \mathrm{fst}\ 0\ y$

2.2. Subterms

Certain concepts are declared as a constant in one proof library but left as a construction over simpler concepts in another. The number 1 can be defined as a single constant *one* (in Proofpower), by the successor of zero $S(0)$ in all the libraries that use a unary representation of numbers or by a binary representation (for examle $BIT_1(0)$). In some cases it is only clear from the context, whether a certain subterm is supposed to represent a single constant or a more complex construction. Consider the HOL4 type $prod(real, real)$. It is used to represent the complex numbers as well as pair of reals (Gauthier and Kaliszyk, 2015).

Matching of whole subterms also enables us to automatically factor type arguments. Type arguments are usually the first arguments of a function, therefore thanks to currification we can find subterms that represent type instances of constants. This approach works even with the type classes of Isabelle/HOL. For instance, in the experiments we will discover that the subterm *zeronat* in Isabelle/HOL is similar to the constant 0 of type *num* in HOL Light.

Because it is impractical to consider all possible subterms as a concept, we will impose some practical restrictions. First, the selected subterms must be formed from function applications and constants only, in particular they do not contain any variables. Second, we only select a subterm if it appears sufficiently frequently in a proof library. Moreover, a subterm is more likely to represent a concept if it is smaller and the conceptualization of big subterms reduces the complexity of the pattern. So, a simple heuristic is to check whether the subterm appears in a number of theorems greater or equal to two times its size.

Every time a subterm substitution is applicable, we duplicate theorem statements. The substitutions replace the selected subterms by newly defined constants. A simple type inference mechanism on a subterm is used to determine the type of its defined constant, in case types at constant positions are required. We will only use maximal substitutions, where a maximal number of replacements is performed with the one with the largest subterms applied first, since they are the most likely to produce new matching pairs of theorems.

Example. (Running)
By creating a new constant definition $c =_{def} s\ (s\ 0)$, we obtain:

$$\forall y\ x.\ \neg(x = alt\_pi \times c) \vee cos\ x = \text{fst}\ 0\ y$$

Since different substitutions may lead to different pairs, it would be also interesting to consider matching large sets of terms with many possible substitutions. As we do not focus on one representation of terms, the applicability of various term indexing techniques (Graf, 1996; Ramakrishnan et al., 2001; Schulz, 2013) for arbitrary proof assistant terms is left open.

2.2.0.1. Equivalent concepts In an effort to minimize the number of equivalent concepts inside one library, we identify constants that are in the same equivalence class of the equality relation. In practice, it requires recognizing theorems of the forms of $c_1 = c_2$ in order to extract an equality relation and replacing constants of an equivalence class by a new constant representing them. The process of conceptualization of subterms is performed before the construction of the equality relation. Therefore constants representing subterms will also be identified with members of their equivalent class.

Example. (Running)
Relevant equalities found in the library after conceptualization: $2 = c$, $\pi = alt\_pi$.
Substitution by a unique representative of the equivalent class induced by the equalities:

$$\forall y\ x.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y$$

2.3. Associativity and commutativity

Rewriting terms modulo associativity and commutativity in the higher-order setting has been studied by Walukiewicz (1998). The simplifiers of certain proof assistants, including HOL Light and Isabelle, implement procedures for normalizing terms modulo AC as part of their simplifiers. The proof checker Dedukti (Dowek et al., 2003) allows reasoning modulo equations (Blanqui, 2003) which can include AC, therefore contains an algorithm for normalizing $\lambda\Pi$ terms modulo such equations.

The requirements of our algorithm are slightly different than of these above. The usual requirement is to reduce α-equivalent terms to the same normal forms given a number of AC rules and a total ordering $<_{ord}$ on ground terms. This order can be constructed by comparing top constructors and in case of equality recusively comparing subterms. In our context, however, the names of the non-logical constants (and variables) should not influence the term ordering. This is because the properties must be independent of the names of the used constants. Therefore the AC normalization procedure works on abstracted terms where different abstraction symbols are used for constants and types.

The names of constants are abstracted in the theorems, but they are important in the induced pairs of constants. In particular, when both arguments of a commutative constant are α-equivalent, the term ordering cannot compare the two possible orderings of the whole term. Consider for example the theorem statement $g(x) = h(x)$, where g and h are constants and x is a variable. Then in theory we would need to create both versions. As this may be explosive in case of large applications of AC connectives, we have not implemented this yet.

(1) No normalization, even the order of subterms applied to the logical constants is preserved.

(2) Normalization based on AC of the logical constants only. For this ordering:

$$\lambda x.\ (x = \pi \times 2) <_{ord} \lambda x.\ (\pi \times 2 = x)$$
$$\lambda x\ y.\ cos\ x = \text{fst}\ 0\ y <_{ord} \text{fst}\ 0\ y = \lambda x\ y.\ cos\ x$$
$$\lambda x\ y.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y <_{ord} \lambda x\ y.\ cos\ x = \text{fst}\ 0\ \vee \neg(x = \pi \times 2)$$
$$\forall x\ y.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y <_{ord} \forall y\ x.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y$$

We get by applying rewrites starting from the deeper subtrees (innermost):

$$\forall x\ y.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y$$

(3) The set of all constants that have an associative or a commutative property in the corresponding proof library. This become imprecise in higher-order foundations, where AC properties may be stated using higher-order predicates (for example $associative(+)$). We get by applying rewrites starting from the deeper (innermost) subtrees: The constant $\times$ is commutative, but since constants are abstracted the ordering cannot distinguish between $\pi \times 2$ and $2 \times \pi$. So the running example is left unchanged.

(4) Finally we add the commutativity of all constants. This means that given a function, the procedure reorders its arguments (possibly more than 2) so that the resulting term is minimal in the term ordering. If $\lambda y.\ fst\ y\ 0 <_{ord} \lambda y.\ fst\ 0\ y$ then the normalized form of the running example becomes:

$$\forall x\ y.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ y\ 0$$

The last normalization may seem strange at first as it creates an inconsistent normalization. However, it does allow for matching concepts which are stated with differently ordered arguments in different libraries.

2.4. Typing Information

In order to find more matches, we also try to normalize the type information across the different proof libraries. We consider four levels of typing available before term normalization and we depict their effect on the following example.

Example. (Typing) In this statement, the constants *hd*, *list*, *int* and 0 respectively stand for head, type constructor for lists, integer and zero.

$$\exists\ l : list\ int.\ hd\ int\ l = 0$$

(1) Type erasure. The type matches can be recovered using the types of the constants involved in a matching and applying type coherence (see Section 5.1).

$$\exists\ l.\ hd\ int\ l = 0$$

(2) Simple types. We create a simple type (one constant) for each unique formula occurring on the left of a type judgment. This approach can be useful if we consider their types, for example when matching constants with dependent types of Coq against set theory constants typed using the Mizar soft type system. Given a new constant definition $d =_{def} list\ int$, the typing example normalizes to:

$$\exists\ l : d.\ hd\ int\ l = 0$$

(3) Variable types. Including the types of all variables is enough to recover all types in simple type theory; however it is not enough to recover all types in more intricate type systems.

$$\exists\ l : list\ int.\ hd\ int\ l = 0$$

(4) Constant types. This combines the previous approach with the types of constant at each positions inside the terms. In this last typing example, \$t is written t for simplicity.

$$\exists\ l : (list : t \Rightarrow t)\ (int{:}t).\ (hd : (\forall a{:}t.\ (list : t \Rightarrow t)\ a \Rightarrow a))\ (int{:}t)\ l = (0{:}(int{:}t))$$

The proposed type levels of typing are recursive, which means that the types are themselves constants whose types are also included in the formula until a defined type, including the basic types of propositions \$*o* and types \$*t*, is reached. The later typing levels are available only to the proof libraries where they are meaningful. In our case study, this implies that the fourth typing information level is not available for Mizar. In Mizar's soft type system a term does not have a unique type, and checking whether a term belongs to a type does require theorem proving. It would be possible to make use

of the cluster-rounding algorithms implemented by the Mizar checker (Trybulec, 2007), even so with terms belonging to many types, this would not match to any of the other considered proof libraries so far.

Using the different levels of typing information increases the accuracy of patterns and allows the use of more precise settings, which limit the number of ambiguous matches. However, additional typing information increases the number of missed matches, that would require type alignments not detected by our algorithm.

3. Similarity

Pairs of constants have an intrinsic similarity based on the number of properties they share as well as the quality of the theorem pairs that created those properties. Various heuristics helps us value the similarity of these pairs of theorems. The most important heuristic for a pair of theorems is the quality of the induced pairs of constants. The correlations between pairs of concepts is captured by our scoring functions. Applying these functions iteratively result in a dynamical system, where the confidence on a pair of concepts evolves relative to the influence of the other pairs. The propagation of this effect stops when the system reaches a stable state, which it always does as demonstrated in Section 3.6. When a stable state is reached, the similarity between concepts should correspond to inherited higher scores.

3.1. Sets of pairs

Before we introduce concrete pair scoring functions, we discuss how the pairs are computed in order to explain the motivation for the functions.

Given two proof libraries lib_1 and lib_2, we first compute all the properties. We next create all possible theorem pairs, by considering all the properties which appear in both libraries. For each property P we apply the same pairing mechanism. Given n_1 theorems from the library lib_1 representing this property, and n_2 theorems from the library lib_2 which represent the property P, we create $n_1 * n_2$ theorem pairs by considering the Cartesian product of the two sets. In practice, the number of theorems sharing a property is small compared to the size of the library(see Section 4.2). Therefore the observed time complexity is less than quadratic with respect to the number of theorems in the library.

The list of all pairs of theorems is named $(t_w)_{1\leq w\leq m}$.

We then compute the set of induced pairs of constants from each pair of theorems. The union of these sets is the set of all pairs of constants for our library pair (lib_1, lib_2). The list of all pair of constants is named $(c_v)_{1\leq v\leq n}$.

Remark. Notice that the variables t and c stand for pairs of theorems and constants and not single theorems and constants. We will rarely need to access the components of the pairs, and we will explicitly refer to the first and second component of the pairs in such cases.

3.2. Scores

We define the scores of pairs of objects recursively by:

$$score_t(t_w) = coef_t(t_w) \times \sum_{i=1}^{n} \delta(c_i, t_w) score_c(c_i)$$

$$score_c(c_v) = g(coef_c(c_v) \times \sum_{j=1}^{m} \delta(c_v, t_j) score_t(t_j))$$

where δ is the characteristic function of the relation R "is induced by" defined in Section 2:

$$\delta(c, t) = 1 \text{ if } c \ R \ t \text{ and } 0 \text{ otherwise,}$$

g is a normalization function from $\mathbb{R}^+$ to $[0; 1[$:

$$g(x) = \frac{x}{x+1}$$

and $coef_t(t_w)$,$coef_c(c_v)$ are coefficients based on heuristics defined and justified in Section 3.3.

In the following, we will use *score* and *coef* when it is clear from the context which of the scores and coefficients are meant.

The function g guarantees the convergence of the vector sequence generated by the algorithm (see Section 3.6). It also reduces the difference between good and very good values providing a smoothing of the scores. This function has a similar role as the sigmoid in backpropagation neural networks (Hecht-Nielsen, 1989).

3.3. Heuristics

The principal reasons for the choice of the following heuristics for coefficients is simplicity, trials and errors, and inspiration from the TF-IDF (Jones, 1972) heuristics for finding the most relevant words in a text. Our heuristics include the coefficients for the *score* functions that scale the sum of the scores of their dependent objects (scores of theorems depends on score of constants and conversely). The coefficient for theorem pairs is composed of two parts, where the first part estimates the quality of the property represented by the pair and the second part corresponds to the recursively considered constants.

In order to estimate the quality of a property, we first compute the frequency of the property $P(t_w)$ of the pair t_w in the set of pairs of theorems, i.e. the number of pairs of theorems representing the property $P(t_w)$:

$$freq(t_w) = card\{t'_w \mid P(t_w) = P(t'_w)\}$$

Since rarer properties should get a higher score we compose this evaluation with a decreasing function.

$$inv\_freq\_property(t_w) = \frac{1}{ln(2 + freq(t_w))}$$

where the property $P(t)$ of a pair t is the property of its first component, which is also equal to the property of its second component.

The second part of the estimation is based on the constants used in the computation of the theorem pair score:

$$ind(t_w) = card\{c \mid c \ R \ t_w\}$$

$$inv\_ind(t_w) = \frac{1}{ln(2 + ind(t_w))}$$

$$average\_const\_score(t_w) = inv\_ind(t_w) \times \sum_{i=1}^{n} \delta(c_i, t_w) score_c(c_i)$$

Remark. If we consider scores as probabilities, it would seem natural to take a product instead of a sum of these probabilities. However, a general guiding rule is that theorem pairs should benefit from an additional good matches in its constants pairs, and be penalized by additional bad matches. There are two possibilities of enforcing such constraints. Either we can take the sum of the scores (which we chose to do) or we can allow the product of the scores to have a value greater than 1. Given the guiding rule, the simple interpretation by probabilities does not create a representative model for our scores.

The total score of a theorem can also be computed by multiplying the inverse of the property frequency by the average constant score:

$$score(t_w) = inv\_freq\_property(t_w) \times average\_const\_score(t_w)$$

This implies that the coefficient of the theorem pair t_w is:

$$coef(t_w) = inv\_freq\_propert(t_w) \times inv\_ind(t_w)$$

We will next give the coefficient for constant scores. If two constants appear in many theorems, they are more likely to have some common properties, whereas rare constants with the same amount of properties should be advantaged. Therefore we apply the following coefficient to the scores of pairs of constants.

$$freq(d) = card\{\text{theorems containing the constant } d\}$$
$$inv\_freq\_const(c_w) = coef_c((d_1, d_2)) = \frac{1}{ln(2 + freq(d_1) \times freq(d_2))}$$

where d_1 and d_2 are the components of the pairs c_w.

This coefficient is the only part of the scoring function which requires inspecting the constants composing a pair.

The *inv_freq_property* is definitely the most important of these coefficients, which together with the correlations created by the sums comprises the core of the algorithm. The other two *inv_ind* and *inv_freq_const* are not critical.

3.4. A dynamical system

In this section we will discuss the relation between our algorithm and dynamical systems, which are algorithms that iteratively refine the interrelated scores. Our algorithm starts by assigning the value 1 for each pair of constants. Next, it calculates a score for each pair of theorems which in turn gives a new score to the pair of constants. This process is then repeated until a fixpoint is reached.

Our experiments will confirm the effectiveness of this approach, however first we will present a theoretical point of view, where we consider the iterative process as a dynamical system. This will reveal more about how pairs of concepts are connected. And the study of properties such as convergence and regional uniqueness, will give us the assurances about the termination of the process and its sensitivity to initial conditions. Finally some important but non-critical conjectures will be made for a global uniqueness and the rate of convergence.

In order to distinguish the different steps of the algorithm we will add the second argument to the score functions, referring to the scores at time t as $score(x, t)$. We can now express the scores of each pairs of concepts at time $t + 1$ in function of the pairs of each concepts at time t. In the following, $(X^t)_{t\in\mathbb{N}} = (x_i^t)_{1\leq i\leq n} \in \mathbb{R}^{+n}$ stands for the

series of successive vectors of scores of constants. Since we can express the scores of pairs of theorems in function of these vectors, we derive the following relation between scores of constants at time $t+1$ and t:

$$x_v^{t+1} =_{def} score_c(c_v, t+1) = g(\sum_{j=1}^{m} l_{v,j} score_t(t_j, t))$$

where

$$l_{v,j} = coef_c(c_v)\delta(c_v, t_j)$$

Similarily

$$score(t_j, t) = \sum_{i=1}^{n} k_{j,i} x_i^t$$

where

$$k_{j,i} = coef(t_j)\delta(c_i, t_j)$$

By linearity we obtain:

$$x_v^{t+1} = g(\sum_{j=1}^{m}(\sum_{i=1}^{n} l_{v,j}k_{j,i}x_i^t)) = g(\sum_{i=1}^{n}(\sum_{j=1}^{m} l_{v,j}k_{j,i}x_i^t)) = g(\sum_{i=1}^{n}(\sum_{j=1}^{m} l_{v,j}k_{j,i})x_i^t)$$

Given the coefficients $a_{v,i} = \sum_{j=1}^{m} l_{v,j}k_{j,i}$ we have:

$$x_v^{t+1} = g(\sum_{i=1}^{n} a_{v,i}x_i^t) =_{def} f_v(x_0^t, \ldots, x_n^t)$$

Essentially, each component at time $t+1$ is given by a linear function of the components at time t combined with the normalization function g. We name this combined function function f_v. We denote by f the compound function defined by $X^{t+1} = f(X^t)$. The linear part $A = (a_{v,i})_{1 \leq v,i \leq n}$ of f is called the correlation matrix of our system.

Remark. We restrict our study to non-negative scores, since f is stable in $\mathbb{R}^{+n}$ and the starting value is in $\mathbb{R}^{+n}$.

Famous examples of dynamical system include the Mandelbrot fractal which reveals the complexity of determining convergence regions, the double rod pendulum and the Lorentz attractor which show extreme sensibility to the initial conditions.

The theory of dynamical systems appears naturally in many scientific domains. Various parameters that affect the stability of such system are presented by Barbarossa (2011) in order to understand biological mechanisms. The relationship between dynamical systems and Markov chains, hinted by the correlation matrix that resembles a transition matrix, was studied by Attal (2010) in the context of open physical systems.

Our system was designed to be a discrete strongly-monotone multi-dimensional non-linear dynamical system. This implies that our system is non-chaotic and posseses many other general properties of strongly-monotone systems, Hirsch (1988).

3.5. Correlations

We will now show how scores interacts with each other. Two pairs of constants are correlated if they are induced by the same pair of theorems. The number and quality of those pairs of theorems decide the strength of the correlation. This is measured by the

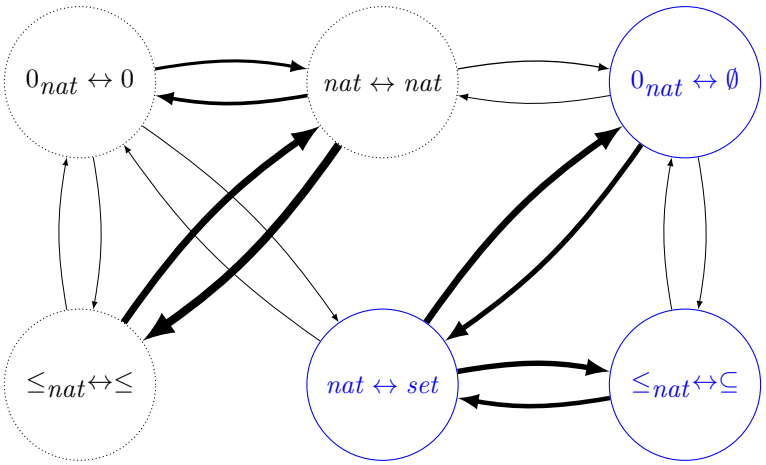

Fig. 2. Part of the correlation graph of Isabelle/HOL-Mizar matches with stronger correlations drawn with wider arrows.

coefficients $a_{v,i}$ of the correlation matrix (see Section 3.4). In Fig 2, a graph representing a part of the correlation matrix created by aligning Isabelle/HOL with Mizar is depicted. This graph is almost symmetric. The reason for the asymmetry is that the coefficient *inv_freq_const* gives a small penalty to pairs having constants appearing in many theorems (see Section 3.3). Thus the influence received by a pair with rarer constants is slightly stronger. A time lapse representation of the dynamic scoring reveals how it is affected by correlations in Fig 3. It demonstrates that scores continuously update by taking into account changes in the other pairs through those correlations.

3.6. Soundness of the algorithm

To be confident that our algorithm terminates and to determinate if the choice of the initial of the arbitrary conditions influences the result of our algorithm, we prove some properties of stability of the dynamical system namely convergence and regional uniqueness.

3.6.1. Proof of convergence

Theorem 3 (Bounded property)**.** *The image of f is in $[0;1[^n$.*

Proof. The image of g is in $[0;1[$. Therefore the image of each f_v is in $[0;1[$ because $a_{v,i} \in \mathbb{R}^+$. The thesis follows. □

Definition 4 (Less or equal)**.** Let $X = (x_i)_{1 \leq i \leq n}$ and $Y = (y_i)_{1 \leq i \leq n}$ be in $\mathbb{R}^{+n}$.

$$X \leq Y \Leftrightarrow_{def} \forall i \in [\![1;n]\!].\ x_i \leq y_i$$

Remark. This is a partial order.

Definition 5 (Increasing)**.**
$X = (x_i)_{1 \leq i \leq n} \in \mathbb{R}^{+n}$ and $Y = (y_i)_{1 \leq i \leq n} \in \mathbb{R}^{+n}$.
A function $f : \mathbb{R}^{+n} \to \mathbb{R}^{+m}$ is increasing when:

$$X \leq Y \Rightarrow f(X) \leq f(Y)$$

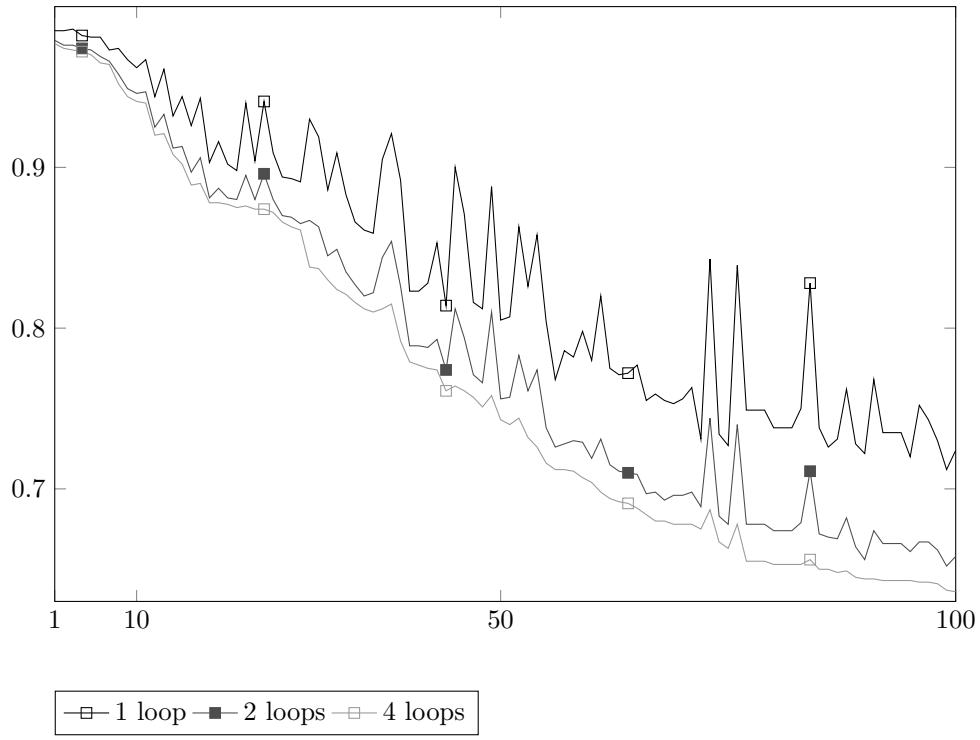

Fig. 3. Scores of the best 100 constant pairs when matching Coq with HOL4 after 1,2,4 loops, ordered by their rank after 16 loops. The graph after 16 iterations is not presented here as it would be very close to the 4 loops curve.

Theorem 6 (Increasing property)**.** *f is increasing.*

Proof. The function g is increasing and $a_{v,i} \in \mathbb{R}^+$, therefore each function f_v is increasing. The thesis follows. □

Theorem 7 (Existence)**.**

$$(X^0 \in \mathbb{R}^{+n} \wedge X^0 \leq X^1) \Rightarrow (\exists X^{lim}.\ \lim_{t\to\infty} X^t = X^{lim})$$

Proof. By induction. We have $X^0 \leq X^1$ by assumption. Suppose $X^t \leq X^{t+1}$. Thus by the increasing property, $X^{t+1} = f(X^t) \leq f(X^{t+1}) = X^{t+2}$. Therefore, the sequence $(X^t)_{t\in\mathbb{N}}$ is increasing.

Each function f_v restricted to non-negative real numbers has its image in $[0;1[$. By an easy induction using the bounded property, each component of $X = (x_i)_{1\leq i\leq n} \in \mathbb{R}^n$ and $Y = (y_i)_{1\leq i\leq n} \in \mathbb{R}^n$ is increasing and bounded in $\mathbb{R}$ which implies that each of them converges and so does the full sequence. □

Theorem 8 (Existence: decreasing case)**.**

$$(X^0 \in \mathbb{R}^{+n} \wedge X^0 \geq X^1) \Rightarrow (\exists X^{lim}.\ \lim_{t\to\infty} X^t = X^{lim})$$

Proof. Analogous to the previous theorem. □

Remark. Although $X^0 \leq X^1$ (or $X^0 \geq X^1$) is only a sufficient condition, there exist sequences that do not converge if this assumption is omitted. 2-cycles are examples of such sequences.

Corollary 1. The point $I^0 = (1, \ldots, 1)$ is converging.

Proof. The image I^1 of I^0 is in $[0;1[^n$ thus $I^1 < I^0$. □

Theorem 9 (Regional uniqueness)**.** *If there exists a sequence* $(X^t)_{t\in\mathbb{N}}$ *such that*

$$\lim_{t\to\infty} X^t = X^{lim}$$

then

$$\forall Y.\ X^{lim} \leq Y \leq X^0 \Rightarrow \lim_{t\to\infty} f^t(Y) = X^{lim}.$$

Proof. Since f^t is increasing and X^{lim} is a fixpoint for f:

$$f^t(X^{lim}) \leq f^t(Y^0) \leq f^t(X^0)$$

$$X^{lim} \leq Y^t \leq X^t$$

By the squeeze theorem on each component $\lim_{t\to\infty} Y^t = X^{lim}$. □

Corollary 2. All points in the higher-dimensional rectangle defined by I^{lim} and I^0 converge to the same limit:

$$\forall Y.\ (I^{lim} \leq Y \leq I^0 \Rightarrow Y^{lim} = \lim_{t\to\infty} Y^t = I^{lim})$$

Remark. All previous results still hold if we replace f by a function f' provided that it is defined on $\mathbb{R}^{+n}$, increasing and its image is in $[0;1[^n$.

Remark. There are at most 2^n fixpoints because a simple substitution of variables creates a polynomial in one variable of degree at most 2^n. This occurs for example when all components are independent of each others. Indeed, each equation can have 2 solutions which implies 2^n fixpoints.

The uniqueness of the attracting point is important, because it ensures that for almost all starting values in $\mathbb{R}^{+*n}$, the final scores are the same. It is easy to prove that is true for $n = 1$ and we conjecture that it is true in general.

Hypothesis 1 (Global uniqueness)**.** Almost all points in $\mathbb{R}^{+n}$ converge to a unique fixpoint in $\mathbb{R}^{+n}$.

3.6.2. Rate of convergence

To estimate the rate of convergence, we need to look at the eigenvalues of the linear operator l that approximates our differentiable function f near the fixpoint reached by I_0. These eigenvalues determine the structure of the phase space and may yield very different outcomes depending on the parameters (Barbarossa, 2011). Assuming the global uniqueness property, the fixpoint is locally attractive which implies the following conjecture.

Hypothesis 2 (Local stability)**.** All eigenvalues of the linear operator l have module less than 1.

A consequence of this property for our algorithm is that it converges linearly in the region R with an error bounded by r^n where r is a positive real smaller than 1. Therefore the scores computed at each iteration become more precise. In our experiments we will stop the recursion when the difference between the scores in two steps is below 0.001. The results in Table 5 shows that it takes about 30 iterations to reach such state.

3.7. Translation: scoring substitutions

Since translations is one of the most important application of aligning libraries, we give here a small preview of how it would work.

To illustrate this, we take the same example Fig 2. Let us suppose that we would like to translate a theorem containing the constants 0_{nat},$\leq_{nat}$ and *nat* from Isabelle/HOL to Mizar. Our objective will then be to find an counterpart for each of the constant in the theorem in Mizar. We will call such the set of such mappings a substitution. Since there are multiple possibilities for each counterpart, the number of possible substitutions grows exponentially. Therefore, we need to figure out which ones are the best.

A simple method is to just take the highest scoring pairs involving those constants. However, it can lead to an incoherent translation since each match is considered independently. In our example, it may translate *nat* to *nat* but 0 to the $\emptyset$. That is why it is important to consider the correlations between matches inside a substitution. A possible method would be to find the substitutions that have the smallest diameter in the graph, choosing the distance to be the inverse of the correlation. In our example, the top two scoring substitutions can be extracted from the dotted (black) and non-dotted (blue) sets shown in Fig 2. Those substitutions give us two possible translations. A fully-fledged translation mechanism would rank substitutions based on their correlations and scores. Additionally, it would also check the type of the resulting term.

4. Experiments

All experiments were performed on an Intel Core i5-3230M 2.60GHz laptop with 4 GB memory. The results of all experiments and the source code that produced them are available at:

http://cl-informatik.uibk.ac.at/users/tgauthier/alignments/

We used the following versions of the provers and their libraries. For all provers each "named" theorem was split into its conjuncts as part of the export phase. Further statistics on the libraries are given in Table 1.

	conjuncts of theorems	constants	types	theories
Mizar	51086	6462	2710	1230
Coq	23320	3981	860	390
HOL4	16476	2188	59	126
HOL Light	16191	790	30	68
Isabelle/HOL	14814	1046	30	77
Matita	1712	339	290	101

Table 1. Number of objects in each library. A constant will be considered a type if it appears in the right-hand side of a type judgement. The constants column does not include constants that represent types.

- HOL4 (Slind and Norrish, 2008) version Kananaskis 10. We considered all the theories built as part of the standard build sequence.
- HOL Light (Harrison, 2009) SVN version 225, including the core library built with the system, the standard library (`Library` directory) and the complex and multivariate developments (Harrison, 2013).
- Isabelle (Wenzel et al., 2008) 2016. We used the Complex_Main theory including all the imported theories down to the object logic HOL.
- Coq (Huet and Herbelin, 2014) version 8.5. Recording of the Coq library was performed via a plugin that allowed us to extract the kernel representation of the objects. We considered all theories in the distribution standard library.
- Matita (Asperti et al., 2014) version 0.5.3-1. We used the last released version of Matita that was able to output XML object representation, again considering only the standard library distributed with the system.
- Mizar (Bancerek et al., 2015) version 8.1.03 including the whole Mizar Mathematical Library 5.29.1227. We used the representation processed using the MPTP XML export (Urban, 2006b).

4.1. Logical mappings

Even if the mathematical formulas come from provers with very different foundations, the properties which they represent are similar from a high-level perspective. However, the specifics of each logic make the internal representations (proof assistant kernel representation) quite different. In order to restore the similarity, we transform the logical constructs of each prover to a common representation. The term structure is inspired by simply-typed lambda calculus, where terms are lambda-terms and all logical constructors are constants. We also allow terms to appear on the level of types which enable us to capture various extensions of the logic, including such common ones as polymorphism (for higher-order logic) and dependent types. We perform the following logical mappings adapted to the logic of each proof assistant.

For Coq and Matita which are based on the calculus of construction we perform the following mappings:

- The dependent product construction where the bound variable is not actually used in the body is replaced by an implication, and the used one is identified with universal quantification. For instance, the typing judgment $f : (\forall x : int.\ num)$ can be translated to $(f : int \Rightarrow num)$ because num does not contain x.

	Import	S	Normalization time			Number of properties		
Mizar	15.05	1218	11.22	36.67	50.22	31692	32725	41311
Coq	4.82	414	3.77	8.85	15.30	5517	6717	8686
HOL4	8.07	427	6.45	10.63	23.59	9265	10522	14064
HOL Light	13.80	448	4.69	10.77	25.81	11450	12182	21948
Isabelle/HOL	4.38	444	2.98	5.52	10.84	10521	10953	14729
Matita	0.40	38	0.27	0.51	1.27	1092	1263	1712

Table 2. Effects of different normalization on the number of properties. The two first columns present the import time and number of frequent subterms (S) for each prover. The following columns show the time taken and number of properties produced by different normalization. The levels of normalizations are in order of inclusion: CNF + AC of logical constants, additional type information, additional subterm substitutions.

- The type hierarchy is collapsed to a single type $\$t$ including *Set*.
- The type of propositions *Prop* is represented by $\$o$.
- De Bruijn indexes are replaced by variable names.
- Each of the logical constructs starting with *Case*, *Cast*, *Fix*, *Cofix* and *Letin* are replaced by a fixed logical constant. This makes it impossible to match them with their counterpart in other logics. However, user named theorems do not typically contains these constructs. Indeed, they appear in 397 theorem statements in Coq and 28 theorem statements in Matita.
- A compatible order for the declaration of constants is re-inferred (this is necessary, as we do not record it in the kernel based export).

In HOL4, HOL Light and Isabelle/HOL which are based on higher-order logic, minimal structural modifications are necessary:

- Polymorphic constants are given explicit types arguments, including type variables when they are not instantiated. For example, taking the head of a list $hd\ l$ is rewritten $hd\ a\ l$ where a is the type of the elements of l.
- Explicit types are given to HOL types based on their number of arguments. For instance, the type constructor *pair* that constructs the cartesian product of two types has two arguments so it is given the type $\$t \Rightarrow \$t \Rightarrow \$t$.
- The boolean type which is also the predicate type is mapped to $\$o$.
- The function type operator $>$ is mapped to $\Rightarrow$.

The first-order set theory of Mizar also requires structural changes

- All functions are curried to match their standard higher-order representation. For example: $f(x, y)$ is rewritten as $(f\ x)\ y$.
- A element of type *true* in Mizar is a set as defined by the axioms of set theory. Therefore we map the type *true* to the type $\$t$.

Remark. The logical constants that have been manually mapped are only allowed to match themselves in the algorithm. This restriction is loosened for Mizar where the type $\$t$ is allowed to match any type.

4.2. Most frequent properties

For most provers we normalize formulas by transforming them to CNF and rewriting them modulo AC logical operators. Furthermore, types are included at all constant positions. Only in the case of Mizar, variable types need to be used and more complex types are mapped to unit types represented by a single constant.

The relative size of all libraries can be estimated from the the numbers of theorems presented in Table 1, and the import time shown in Table 2, which corresponds to the total size of the theorems. The average theorem size is small in Coq and Isabelle/HOL, average in HOL4 and large in HOL Light. We also observe that normalization time depends linearly on the theorem size.

The evolution of the number of properties relative to the different normalization is shown in Table 2. The type information increases the precision of the theorems, by splitting equivalent classes of theorems into smaller classes. Conversely, subterm conceptualization relaxes the matching constraints by creating at most one variant for each theorem. Still, we observe that the number of properties grows as new theorems are added to the library.

The most frequent properties for each prover involving one constant are presented in Table 3. Unsurprisingly, commutativity, associativity, and transitivity are very common. In Isabelle/HOL, the ubiquity of type classes reduces the repetition of such properties. On the contrary, many properties are repeated in Coq, with some of them originating from isomorphic structures. These structures are often not identical: some structures lead to more efficient (or relevant) computations whereas others are more convenient for proving. A simple example is the distinction between binary naturals and unary naturals. From a mathematical perspective these two represent the same concept but their algorithmic properties are different.

We also present the most common properties involving two constants in the libraries in Table 4. We manually named the automatically derived properties appearing in Tables 3 and 4 from their term representation. But we could have also rely on a learning-based automated method for naming properties developed by the second author (Aspinall and Kaliszyk, 2016).

4.3. Matching algorithm

In Table 5 we show the performance of the pairing mechanism and dynamical scoring loop included in the matching algorithm. The presented approach is fast and scales across many formal libraries, as opposed to the previously considered approaches (Gauthier and Kaliszyk, 2014). One reason is the efficiency of the pairing step. The other is the limited number of pairs obtained after the pairing step. The scoring loop can then only recurse over these pairs. Assuming, that the numbers of constants inside a property and the number pair of theorems related to a pair of constants are bounded, one iteration of the algorithm is linear in the total number of pairs. This is confirmed by the relatively fast scoring times. Our algorithm converges below the 0.001 threshold in about 30 steps in each pair of provers with no observed dependence on the size of the considered libraries.

The results also give some hints about the similarities between provers. First, the number of theorem pairs can be used as a weak indicator of the degree of similarity between two provers. However it is largely skewed by the size of the libraries. Looking

Coq		HOL4		HOL Light	
Property	Thms	Property	Thms	Property	Thms
Commutativity	157	Injectivity Eq	72	Commutativity	34
Associativity	143	Commutativity	48	Associativity	30
Transitivity	94	Injectivity Eq TA	31	Injectivity Eq	25
Nilpotence	75	Associativity	29	Nilpotence TA	15
Injectivity	63	Transitivity	22	Transitivity	15

Isabelle/HOL		Matita		Mizar	
Property	Thms	Property	Thms	Property	Thms
Injectivity Eq	23	Inductive def	69	Truth 2A	123
Injectivity Eq 2TA	9	Truth	13	Transitivity TA	67
Injectivity Eq 3TA	6	Commutativity	9	Truth 3A	64
Equality def	6	Nilpotence	7	Injectivity	43
Identity def TA	6	Associativity	6	Associativity	41

Property	Pattern
Commutativity	(C0 V0) V1 = (C0 V1) V0
Associativity	(C0 V2) ((C0 V1) V0) = (C0 ((C0 V2) V1)) V0
Transitivity	(C0 V1) V2 ∨ ¬ ((C0 V0) V2 ∨ ¬ ((C0 V1) V0))
Nilpotence	V0 = (C0 V0) V0
Injectivity Eq	(V1 = V0) = (C0 V1 = C0 V0)
Injectivity	¬ (C0 V1 = C0 V0) ∨ (V1 = V0)
Inductive def	(V4 V1) ∨ (∃ V1 V0, (¬ (V4 ((((C0 V3) V2) V1) V0))))
	(V2 V3) ∨ (∃ V1 V0, (¬ (V2 ((C0 V1) V0))))
	(V4 V5) ∨ (∃ V1 V0, (¬ (V4 ((((C0 V3) V2) V1) V0))))
	(V2 V3) ∨ (∃ V1 V0, (¬ (V2 ((C0 V1) V0))))
Equality def	((C0 V2) V2) (λ V1 V0, (V1 = V0))
Identity def	C0 = (λ V0, V0)
Truth	C0
Truth 2A	C0 V1 V0

Table 3. Most frequent properties involving one constant in number of theorems. The suffix "xA" precises the number of arguments "x" of the constant. The suffix "xTA" precises the number of silent arguments (often type arguments) "x" of the constant. The property "Inductive def" actually regroups four different properties that are abstracted from inductive definitions.

at the number of common properties is slightly more convincing, as shown by the 1457 common properties shared by HOL Light and HOL4.

Coq		HOL4		HOL Light	
Property	Thms	Property	Thms	Property	Thms
Left distributivity	256	Inverse F	58	Implication 2A	95
Right distributivity	169	Linearity	43	Property on	33
Left neutral	133	Different	41	Monotonicity	28

Isabelle/HOL		Matita		Mizar	
Property	Thms	Property	Thms	Property	Thms
Inverse F	19	Implication	18	Right neutral F	101
Implication 3A	18	Inverse F	10	Left distributivity	80
Implication 2A	11	Structure of	9	Inverse F	74

Property	Pattern
Left distributivity	(C0 ((C1 V2) V1)) V0 = (C1 ((C0 V2) V0)) ((C0 V1) V0)
Right distributivity	(C0 V1) ((C1 V2) V0) = (C1 ((C0 V1) V2)) ((C0 V1) V0)
Left neutral	V0 = (C1 V0) C0
Right neutral F	C1 V0 = C1 (C0 V0)
Inverse F	V0 = C1 (C0 V0)
Linearity	C0 ((C1 V1) V0) = (C1 (C0 V1)) (C0 V0)
Monotonicity	(C1 V1) V0 = (C1 (C0 V1)) (C0 V0)
Implication	C1 ∨ ¬ C0
Implication 2A	(C1 V1) V0 ∨ ¬ (C0 V1) V0
Implication 3A	((C1 V2) V1) V0 ∨ ¬ ((C0 V2) V1) V0
Different	¬ (C1 = C0)
Property on	C0 (λ V3, ((C1 (V2 V3)) (V1 V3))) V0
	∨ ¬ ((C0 V2) V0) ∨ ¬ ((C0 V1) V0)
Structure of	C1 | ¬ (C0 V0)

Table 4. Most frequent properties involving two constants in number of theorems. The suffix "xA" precises the number of arguments "x" of each constant. The suffix "F" precises that the property should be understood at the function level.

To give better estimates, we further base our analysis on the scores of the pairs of constants. Each of the subsequent graphs reproduce the scores of the best matches, when aligning two libraries.

4.4. Effect of normalization

According to the heuristics presented in Section 3.3, there are two main ways normalization can improve the score of a match. First, by creating more theorems pairs that induces this match. This will in practice imply that the constants involved in the match

	Pairing	Props	T pairs	C pairs	Scoring	Loops
Coq - Mizar	1.74	833	46113	34772	13.43	38
HOL4 - Mizar	1.48	814	36918	26433	10.9	36
Coq - HOL4	0.92	500	32127	15365	7.37	33
HOL Light - Mizar	1.34	679	27395	20085	6.65	29
Coq - HOL Light	0.52	379	24600	11189	5.78	34
Isabelle/HOL - Mizar	0.92	558	20363	13838	5.99	35
Coq - Isabelle/HOL	0.38	349	19758	8273	2.35	19
HOL4 - HOL Light	0.44	1457	18296	5861	2.68	23
Coq - Matita	0.19	250	13365	5147	2.65	34
HOL4 - Isabelle/HOL	0.21	427	10074	4562	2.06	32
Matita - Mizar	0.32	221	9401	4549	1.48	21
HOL Light - Isabelle/HOL	0.20	392	7552	3208	1.41	30
HOL4 - Matita	0.10	158	3469	1434	0.70	29
HOL Light - Matita	0.14	120	2335	1043	0.50	28
Isabelle/HOL - Matita	0.08	117	1540	962	0.50	32

Table 5. Statistics for each pair of provers gathered during pairing and dynamical scoring of pairs of constants and theorems, ordered by their number of pairs of theorems. Presented form left to right in this table: pairing time (in seconds), number of common properties, number of theorem pairs, number of constant pairs, scoring time (in seconds) and number of loops necessary to reach a fixpoint.

will share more properties. Second, by decreasing the frequency of the theorem pairs as their precision is increased. Each normalization step has a positive and a negative effect on the score. Subterm conceptualization, CNF normalization, and AC rewriting increase the numbers of common properties but diminish their accuracy. The reverse is true for typing information. The combination of those effects is presented in Fig 4. Aligning Coq and Mizar, the positive influence prevails in all cases. The addition of type information contributes to the largest improvement. Aligning HOL Light and Isabelle/HOL, subterm conceptualization is the game changer. Indeed, automatic factorization of type arguments is made possible through subterm conceptualization. This is essential for aligning other provers with Isabelle/HOL because type arguments occurs naturally during the instantiation of type classes. For the two considered pairs of provers, the application of commutativity to all operators improves the scores minimally. It is not clear whether this minimal score gain translates into more accurate matches, therefore we will not include this normalization by default.

4.5. Evaluation of the best scoring pairs

The matching algorithm was run on all pairs of provers and the scores of the first thousand best matches are depicted in Fig 5. The HOL4 and HOL Light provers share many similar concepts. The alignment of Matita is significantly better with Coq than with any other provers. Since the library of Coq contains a lot of basic mathematical and algebraic concepts it can be aligned well with a variety of provers.

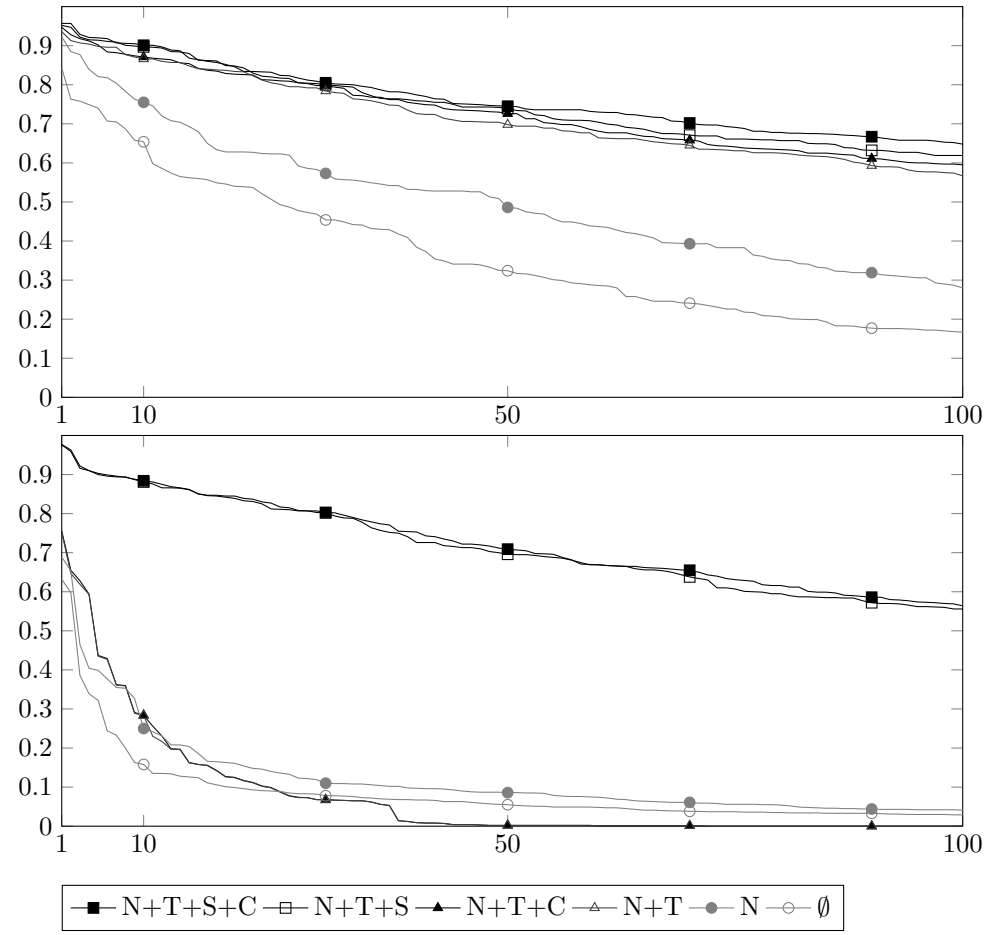

Fig. 4. Effect of normalization when aligning Coq with Mizar (top figure) and HOL Light with Isabelle/HOL (bottom figure). "N" stands for CNF normalization + logical AC. "T" stands for default typing formation. "S" stands for subterm conceptualization. "C" stands for the fourth level of AC normalization which includes permutation of arguments in non-commutative constants.

The presented scores only give an estimate of the quality of the matches. For efficiency reasons, our algorithm only focuses on a syntactic analysis of the similarities. Semantic measures are needed to capture the full complexity of a concept. More, repetition inside one library may artificially increase the number of concepts with strong matches. Therefore, a manual inspection of the pairs is necessary to evaluate the final quality of the matches.

To reduce the number of manual inspections to a reasonable level, we will focus our human analysis on the following pairs of provers by order of subjective difficulties: HOL Light-HOL4, HOL Light-Isabelle/HOL, HOL4-Isabelle/HOL, Coq-Matita Coq-HOL4, Isabelle/HOL-Mizar, and Coq-Mizar. To further simplify our analysis, we make a distinction between three classes of matches. Because it is a subjective judgment, there is no strict limit between each of the following classes.

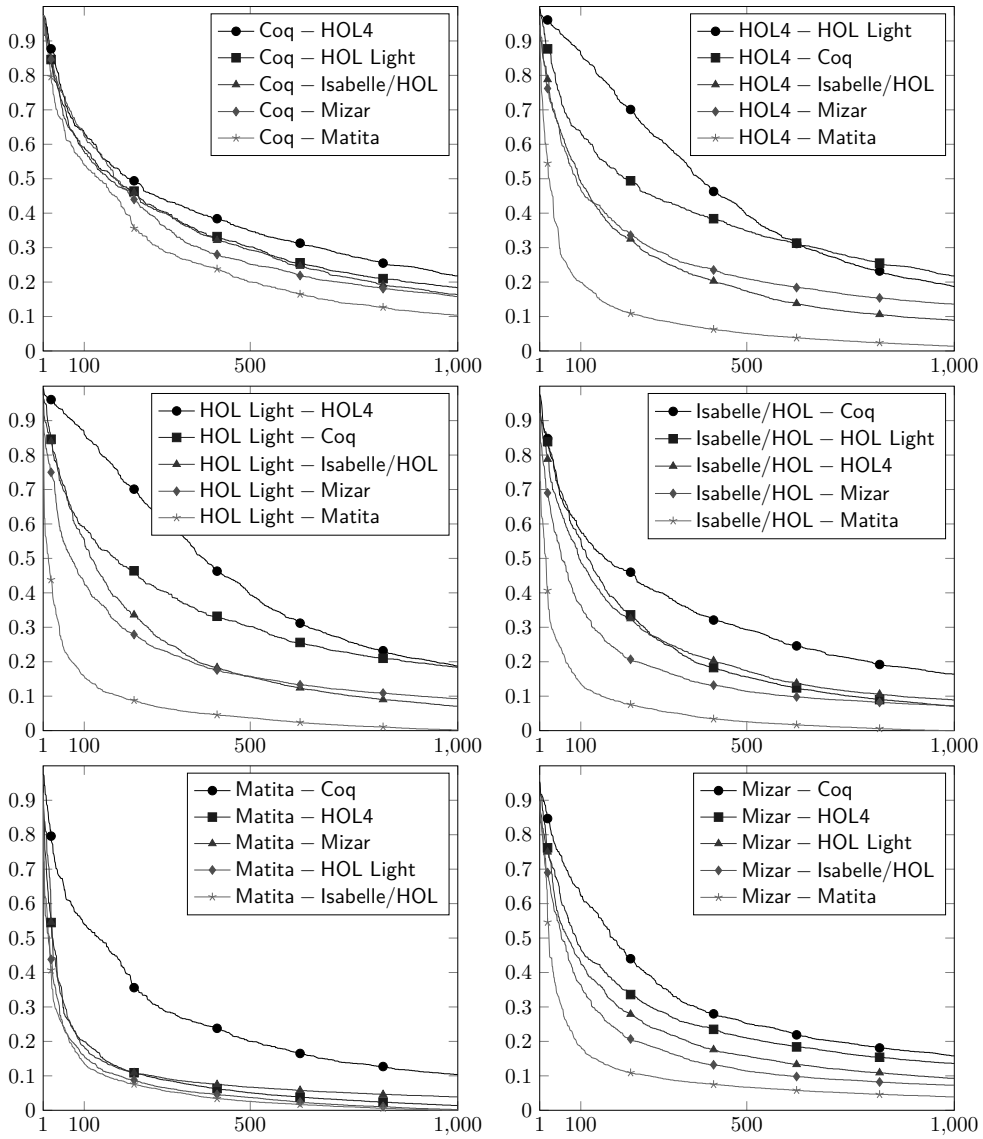

Fig. 5. Scores of the best thousand pairs of constants with default settings.

A pair of constant will be called:

- an optimal match, if the two concepts were intended to represent the same mathematical object, although their definitions may differ. A match between a polymorphic constant with an implicit type argument and one of its instantiations will also be considered optimal.
  Examples of such optimal matches include: reals numbers defined differently, binary and unary natural numbers, specific instance of addition and addition ($+_{int}$ vs $+$).
- an approximate match, if its components are the carriers or the operators of a known morphism between large mathematical structures.

Example: sets vs list of lists, reals vs integers, $<_{real}$ vs $<_{int}$, union vs intersection.

- a singular match if it comes from a smaller morphism appearing inside a structure. Example: division vs less than, $+_{real}$ vs $*_{int}$.

Depending on the application, the range of interesting matches may differ. For a translation between libraries recognition of optimal matches is necessary, whereas to take inspiration from proofs in other domains approximate and singular matches are also of high value (Gauthier and Kaliszyk, 2015). The addition of high-scoring singular and approximate pairs will be useful to learn patterns in the use of these concepts and in their relation to proving.

Overall, our matching algorithm tries to evaluate the quality of the mapping in the context of the two whole libraries. Therefore optimal matches are expected to have higher scores, followed by approximates ones and finally singular ones. The degree to which our algorithm is capable of ordering the matches correctly is presented in Table 4.5.

Among the provers of the HOL family the first non-optimal match occurs quite early but is always preceded by a conflicting optimal match. As there are six different definitions of natural numbers in Coq, our algorithm has to find that these six competing versions match the HOL Light natural numbers. Moreover the relation between algebraic structures is lost. Indeed, the reals from Coq are constructed from only one specific version of the numerals.

Similar issues occur when aligning Mizar with any of the other provers. The issues are further amplified by the fact that many constants in Mizar are implicitly polymorphic. Combining the two effects, + in Mizar matches to the different version of + for integers in Coq but also + for integers, complex numbers and reals. Since most of the the first hundred matches are of this kind, we have to look further for matches about more involved concepts, such as lists and trigonometry. This means that many approximate matches and a few singular matches are mixed with the optimal ones. We will therefore define additional methods to separate them in Section 5.

4.6. Transitive matches

When experimenting with more than two libraries, it is possible to consider one library as a translation step between two others. Indeed, concepts can be mapped from the initial library to the intermediate library and then to the targeted library. We will give a transitive score *transitive_score* to such matches defined as the product of the intermediate scores on the path linking the two constants. In the following experiment we will look at the transitive matches produced when our initial and final libraries are Isabelle/HOL and Mizar (the order is irrelevant) and the other libraries are used as intermediates. We only map concepts through one intermediate library at a time in order to measure their performance. This approach can easily be generalized so that the mapping travels through multiple translators.

In Fig 6, the two best intermediates seem to be Coq and HOL Light followed by HOL4. However, because of the multiplication of similar structures in Coq, most interesting matches will be found in HOL Light. Compared to direct scores, transitive matches may have an unfair advantage as they can generate a large number of one-to-many mappings.

In order to evaluate the gain obtained with the help of transitive matches, we compare the transitive scores to the scores (*direct_score*) that were attributed during a direct

Prover 1	Prover 2	Constant 1	Constant 2	rank	score
HOL4	HOL Light	**num**	**real**	25	0.96
		num	*num*	2	0.99
		real	*real*	1	0.99
HOL4	Isabelle/HOL	**real**	**nat**	4	0.89
		real	*real*	2	0.95
		num	*nat*	1	0.97
HOL Light	Isabelle/HOL	**real**	**int**	10	0.88
		real	*real*	1	0.97
		int	*int*	24	0.81
Coq	Matita	**Z**	**nat**	1	0.97
		Z	*Z*	12	0.87
		nat	*nat*	2	0.97
Coq	HOL4	**Z**	**real**	2	0.97
		Z	*num*	3	0.97
		R	*real*	10	0.94
Isabelle/HOL	Mizar	**dvd nat**	$\leq$	6	0.82
		dvd nat	*divides*	35	0.58
		less_eq nat	$\leq$	9	0.78
Coq	Mizar	**Z**	**real**	2	0.94
		Z	*integer*	17	0.87
		R	*real*	8	0.91

Table 6. First non-optimal match in each studied pair of provers (in bold), followed by the first optimal matches for each constant

match. We additionally define two other measures to evaluate the novelty of our matches:

$$dif = transitive\_score - direct\_score$$
$$w\_dif = direct\_score * (transitive\_score - direct\_score)$$

Comparing all intermediate libraries, we discover that Coq is actually giving optimal matches of lower quality. In Table 7, the third top scoring match is singular and already the first one uses an approximate intermediate. The matches with best transitive scores through HOL4 and HOL Light are more accurate. To measure the novelty of those matches, we will rely on the other two scores to order them. In Table 9 transitive matches between Isabelle/HOL and Mizar are discovered through HOL Light and ordered by their difference scores *dif*. We first observe that all of the considered matches were not discovered by the direct approach and that the first three are optimal. It means that they were concepts that did not share any property directly but had both properties in common with HOL Light. Using HOL4 (see Table 8), new optimal matches were a bit less frequent. In order to achieve a compromise between novelty and quality we use the scoring function *w_dif*.

Overall, this experiment demonstrates that the use of a transitive method can discover new matches and reinforce the confidence on existing matches. To maximize optimal

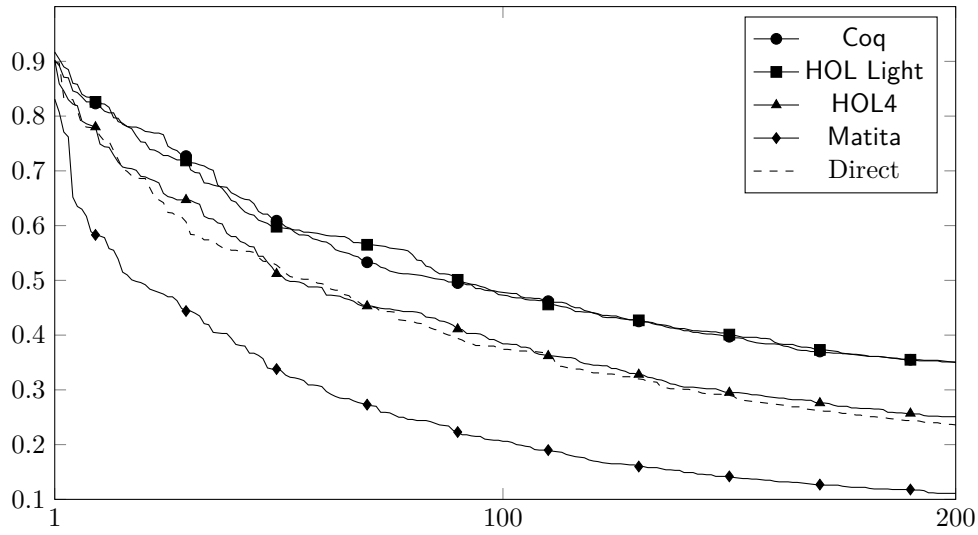

Fig. 6. Best transitive matches between Isabelle/HOL and Mizar.

Isabelle/HOL	Mizar	through Coq	Direct	Transitive
zero int	0	*BinNums_N_0*	0.83	0.82
zero int	0	*BinNums_Z_0*	0.50	0.78
dvd nat	$\leq$	*le*	0.81	0.77
less_eq nat	$\leq$	*le*	0.78	0.77

Table 7. First four transitive matches through Coq (excluding types) with best transitive scores.

Isabelle/HOL	Mizar	through HOL4	Direct	Transitive	*w_dif*
(*less_eq real*)	$\leq$ 0	*real_lte*	0.40	0.64	0.10
(*zero real*)		(*real_of_num* 0)			
cos real	*cos*	*cos*	0.45	0.58	0.06
sin real	*sin*	*sin*	0.45	0.57	0.05
real	*real*	*real*	0.75	0.81	0.04

Table 8. First four transitive matches through HOL4 (excluding types) with best *w_dif* scores.

matches in future applications, a combined score given by the sum of the transitive and direct scores may be considered.

Remark. The transitive type matches were excluded from the tables because the Mizar type system makes the distinction between approximate and optimal type matches fuzzy.

Isabelle/HOL	Mizar	through HOL Light	Direct	Transitive	*dif*
(*less real*)	< 0	*real_le*(*real_of_num*	0	0.50	0.50
(*zero real*)		(*NUMERAL* 0))			
power real	$\|\hat{}$	*real_pow*	0	0.44	0.44
times complex	$*$	*complex_mul*	0	0.40	0.40
pred	*carrier*	*cart real*	0	0.40	0.40

Table 9. First four transitive matches through HOL Light (excluding types) with best *dif* scores.

5. Strategies

In order to further distinguish optimal matches from non-optimal ones, we provide a number of matching strategies. These iterative procedures divide matches into a positive and a negative set. The strategies aim to maximize the number of optimal matches and minimizing the number of singular or approximate matches in the positive set. These two sets are build incrementally. When a pair is chosen to be a member of the positive (respectively negative) category, it receives a positive (respectively negative) reinforcement. The purpose of these reinforcements is to increase or decrease the strength of the influence of all pairs beyond the scores that were attributed in the scoring loop. We first define the terminology used to describe the strategies.

Definition 10 (Positive, negative and undecided matches)**.** A *positive match* is an element of the positive set. A *negative match* is an element of the negative set. An *undecided match* is neither an element of the positive set nor an element of the negative set. It will become positive or negative as the two sets grow.

Definition 11 (Positive reinforcement)**.** A positive reinforcement is a modification applied to the score of a positive match and amounts to:
- Fixing its score to 3.

This number has been experimentally determined and roughly says that we are more than 3 times more confident that a selected match is optimal than for an undecided match.

Definition 12 (Negative reinforcement)**.** From a negative match N, a negative reinforcement performs two modifications:
- Fixing the score of N to 0.
- Fixing the score of each pair of theorems that induces N to 0.

The second modification is justified in Section 5.1 by the constant coherence constraint. This modification would be implicit if we had used a product instead of a sum in the scoring function for pair of theorems. Scores based on products are however not consistent with other constraints as shown in Section 3.3.

All presented strategies first run the dynamical scoring algorithm. Next, a pair (or a set of pairs) of constants is chosen based on their scores and additional heuristics to decide if it (they) should be classified as a positive or as a negative match. A positive (respectively negative) reinforcement is applied to each element of the positive (respectively negative)

set. Dynamical scoring is then rerun to account for the newly updated scores. A new selection is performed, and the whole process is repeated as long as there exist undecided pairs. Pairs with zero scores are always put in the negative set. If not stated otherwise by a strategy, the pair with the highest score will be assigned to the positive set.

Remark. Convergence is guaranteed by the fact that the dynamical scoring algorithm is restarted from fixed initial values greater or equal to 1 after each decision. Reinforcement scores are fixed and non-negative and thus can be treated as coefficients. Reinforcement scores should not be modified by dynamical scoring. Otherwise, the algorithm would always converge to the fixpoint found with no reinforcement by uniqueness of the limit.

Beside a strengthening of the influence of the top matches, another advantage of the two level approach is that we separate algorithms for scoring the degree of isomorphism of matches from the ones deciding which matches are indeed optimal. Therefore, it is possible to use different scoring techniques for each of them, which will be exploited by the disambiguation in Section 5.3. Furthermore, some of the following techniques work globally on optimal matches (and are not helpful for searching for more approximate and singular matches, like it is the case for conjecturing (Gauthier et al., 2016)).

We present three options which can be combined to form a strategy. First, we propose natural coherence constraints on the two set of matches. Second, we consider greedy matching. Third, we discuss disambiguation, which aims at resolving conflicts created by a multiple counterpart mappings. These three options can be used independently or combined and even used together with some human advice. The strongest combinations of the three automatic options is evaluated through the quality of the positive matches they produce.

5.1. Coherence constraints

After correctly identifying a match, simple coherence constraints based on the logical relation between the different objects (types, constants and theorems) can be applied: type coherence and constant coherence.

Definition 13 (Type coherence)**.** A set of constant matches is type coherent if for every constant pair c of a set s, the type matches induced by c are also in the set s.

A weak form of type coherence was already induced by the correlations: Higher scoring constants cause higher scores in the types. However, this influence may not raise the scores of the types enough for it to become a positive match. In some cases other factors of the algorithm might give such types negative advice. To enforce type coherence on the positive set, as soon as we add a match to the positive set, we also add their induced type matches to the positive set.

Remark. Type coherence is recursive, since types have themselves types.

Although we classify constants rather than theorems (this distinction is less pronounced in intuitionistic type theory (Kaliszyk et al., 2014)), we assume in the following definition that the theorems are also assigned to a positive and negative set of theorems. After the strategy terminates, a theorem will be said to be in the positive set if it has a score greater than 0 and in the negative set if its score is 0. We can then state a similar constraint relation between theorems and constants that we call constant coherence.

Definition 14 (Constant coherence)**.** For every theorem pair t of the positive set of theorems, the constant matches induced by t are also in the positive set of constants.

Constant coherence is enforced by zeroing the scores of theorems that contain negative pairs. This constraint also implies that if a type match is in the negative set then the constant pairs that induce it will in the next iteration be in the negative set. Indeed, theorem pairs that include a constant pair will also include its induced type pairs. Therefore if one of its type pairs is given a score of 0, all of the theorem pairs than induce the constant pair are given a score of 0. Hence the total score of the constant will be 0. This consequence can be seen as the contrapositive of type coherence satisfied by the negative set.

5.2. Greedy method

In our dynamical scoring experiments we observe that a match with the highest score involving one constant is most often an optimal one, and the subsequent ones involving the same constant are approximate ones. For example, a match of integers with integers may be followed by a match of integers with reals. In order to remove those undesirable competing matches, as soon as a pair p is categorized as positive, the greedy strategy puts all pairs that have a common constant with p in the negative set. The first advantage of the strategy is a drastic reduction of the number approximate matches. Also, if applied with type coherence and without subterm conceptualization, there will be at most one possible translation of a formula which will type-check by construction. In this particular case, the selection of coherent substitution is not needed (see Section 3.5).

There are however two downsides. First, it makes the algorithm more brittle. The presence of an approximate match early in the procedure leads to a series of approximate matches derived from the first one. For instance, matching integers with naturals leads to all operations about integers being matched to operation about naturals. Second, the algorithm does not allow for one-to-many mappings in the positive set. If there are multiple structures defining the same objects, as it is often the case for example for computational reasons, the greedy algorithm will only map one.

5.3. Disambiguation

Often an approximate match and an optimal match about the same constant are in the wrong order, the scores differing only by a small margin. The greedy method fails to recognize the ambiguity and assumes that the first match is correct.

To solve this problem, we propose a method that delay the selection of a positive match by measuring its ambiguity. To this end, we will penalize a pair if its constants have multiple possible counterparts in the other library. The hope is that the selection of less ambiguous matches will help the classification process.

Technically, we define the ambiguity score of a match $c = \{d_1, d_2\}$ by first defining its ambiguity sets, which are consists of the matches that shares a constant with c.

$$E_1 = \{c' = \{d, e\} \mid d = d_1 \vee e = d_1\}$$
$$E_2 = \{c' = \{d, e\} \mid d = d_2 \vee e = d_2\}$$

And then its ambiguity scores.

$$\mathit{ambiguity}_1(c) = \sum_{c' \in E_1} \mathit{score}(c')$$

$$ambiguity_2(c) = \sum_{c' \in E_2} score(c')$$

$$ambiguity(c) = ln(10 + (1 + ambiguity_1(c)) * (1 + ambiguity_2(c)))$$

$$ambiguity\_score(c) = \frac{1}{ambiguity(c)} \times score(c)$$

Remark. Different ambiguity scores were also tried. Most of them improved the results in a similar manner. Therefore, the quality of matches is not very sensitive to small changes in the scoring functions, and the application of disambiguation is more important than its particular implementation.

It is easy to note that for pairs with similar scores the higher an ambiguity score is the less ambiguous it is. We will then use these ambiguity scores instead for the selection of our positive matches. These new scores will affect our algorithm only by changing the order of the selected positive matches. They will not replace the reinforced scores or the scores of the undecided matches. Thus they do not change the correlation between matches. From our experiments, preventing the ambiguity of one pair from being transmitted to other pairs seems necessary to preserve the stability of dynamical scoring.

5.4. Human advice

Automated techniques may not be sufficiently precise or the user may desire to have more control over the alignment of the two libraries. That is why we also provide ways to combine those techniques with additional human advice.

In this setting, the user is shown the 40 (modifiable) undecided best matches. She can select a few of them and decide which sets they belong to. After the negative and positive reinforcements are applied and scores updated accordingly, the user can repeat the selection on the new 40 undecided best matches. To facilitate the procedure, commands are available to the user that allow him to undo a decision, show the positive and negative sets, display the undecided matches according to different orders and stop the iteration. By default, the order used is defined by the scores. But we also propose to regroup undecided matches by the constants they share (to perform manual disambiguation) or by similarity of the names of their constants. As a compromise, human advice can also be restricted to type matches. In this case, constant pairs (that are not a type pairs) with a score greater than the best undecided type pair are classified automatically.

Those human advice scenarios were not fully evaluated as it was more efficient for us to improve the automation than having us manually compensate for the algorithm's failures. Moreover, since we do not have expert knowledge in all the libraries our decisions were also error-prone. Therefore, these combined methods were mainly useful as debugging tools.

5.5. Results

Experiments are performed on the studied pair of provers. We run two different strategies. Both have type coherence and disambiguation enabled, but one of them applies the greedy restriction where the other one does not. We will refer to them as the greedy and non-greedy strategy.

Prover 1	Prover 2	Sect	Constant 1	Constant 2	rank
HOL4	HOL Light	303	*extreal*	*complex*	227
			extreal_pow	*complex_pow*	228
			extreal_mul	*complex_mul*	229
HOL4	Isabelle/HOL	159	*modu*	*real_norm complex*	39
			prod	*complex*	40
			EVERY	*pred_list*	90
			nub	*remdups*	106
			SNOC	*insert*	107
HOL Light	Isabelle/HOL	123	*FCONS*	*case_nat*	78
			ALL	*pred_list*	79
			DIV	*binomial*	92
			rational	*positive*	108
Coq	Matita	84	N	Z	13
			$= 0_N$	*Zle*	34
			N_mul	*Ztimes*	46
			transitive	*symmetric*	48
Coq	HOL4	188	*rev_append*	*REV*	7
			BinNums_N	*ext_real*	45
			0_BinNums_N	*extreal_of_num* 0	46
			bool	*rat*	49
			constr_bool_2	*rat_1*(*rat_of_num*(*NUMERAL*(*BIT1 ZERO*))	61
			measure	*gtof*	71
Isabelle/HOL	Mizar	137	*list*(X)	*Element* (*QC-WFF*(X))	2
			sup	$\backslash/$	3
			sqrt	$\_^2$	24
Coq	Mizar	168	*RIneq_Rsqr*	*min*	9
			sqrt	$\_^2$	10

Table 10. First suspected non-optimal matches in the positive set. The column Sect shows the total number of elements in the positive set.

5.5.1. Non-optimal matches

We measure the effectiveness of the classification made by the greedy strategy by finding the first non-optimal matches in the positive set. Table 10 shows their rank. It also presents the size of the section which is the total number of matches in the positive set. The small size of each section compared to the total number of matches shows that the strategy is quite effective at eliminating non-optimal matches.

A manual inspection of the definitions in Table 12 reveals that the constants *Rev* and *rev_append* are actually the same. Table 11 also shows how this pair of constants was discovered from their similar properties. It would have been harder to recognize from their definitions. Indeed the first one is constructed by a match statement and the second one by a conjunction. After checking more definitions, we realized that the component of the pairs (*measure*, *gtof*), (*ALL*, *pred_list*) and (*EVERY*, *pred_list*) also represent the

exact same concepts even if their names indicate otherwise. Those misjudged matches could be used to create more consistent naming schemes across formalizations.

The similarity between HOL Light and HOL4 is striking. There exist 790 constants in HOL Light and a little less than 300 of them can be map to a constant in HOL4 that have exactly the same meaning. More, this number does not include the one-to-many mappings discovered by the non-greedy version. Aligning HOL Light and HOL4 to Isabelle/HOL is a bit more challenging. But the algorithm is still effective and the greedy strategy discovered more than 100 optimal matches in each case.

Disambiguation was an essential component for aligning correctly Coq with HOL4. We find that 188 concepts have a counterpart in the other library. These include only a few non-optimal matches, less than 10 in the first hundred matches. Even some of those non-optimal mappings can be interesting. At rank 98 (not shown in the table), the Coq constant *Ensemble*(set in French) is matched with the HOL4 constant *llist*(lazy lists). This match may even appear to be better than an optimal one involving the HOL4 constant *set*, after studying their usage in both provers. In this alignment, the booleans of Coq could not match the related boolean type in HOL4, since it is the same as the reserved type $\$o$ which is restricted to match only to itself. This issue could be solved easily by mapping the boolean type of Coq to $\$o$ during the application of the logical mappings.

The results are more modest for alignments with Mizar, although the size of the positive set is comparable. The percentage of optimal matches decrease rapidly and there are almost no optimal matches after the 50th. This is mainly due to the wrong choices made early on. Therefore, to align a prover with Mizar, it is better to run the non-greedy strategy. Another option, before better logical mappings, normalization or strategies are implemented for Mizar, would be to run the matching algorithm with some human advice. But with a low frequency of correct matches, it would be a tedious manual work and may defeat the purpose of our project.

5.5.2. Optimal matches

We inspect the optimal matches found by the greedy strategy to judge their value. We also compare it to the non-greedy strategy to find which optimal matches the greedy strategy missed. Table 13 presents interesting optimal matches found by the greedy and non-greedy strategies. The selected optimal matches in this table illustrate different achievements of our approach.

Subterm conceptualization renders possible to match pair of reals with complex numbers, and they are indeed used in that way in HOL4. The greedy startegy may however not recognize this similarity. Indeed, the concept of a pair of reals may exist in both prover, yielding the match of the two isomorphic concepts. Therefore it would prevent any further matches involving those concepts. Another effect of conceptualization is the automatic factorization of type arguments. Thus, the subterm *power real* can be identified with the constant *real_pow*. The advantage of regrouping constants using their reflexive transitive closure modulo equality appears when the two constants *PI* and *ALT_PI* automatically match the same counterpart *P_t*. The concepts *relation*, *decidable*, *transitive* shared by Matita and Coq show the different degree of abstractions of each library. We can also illustrate the effectiveness of type coherence when aligning Coq and HOL4. A match between the constants *length* and *LENGTH* (representing the length of a list) directly implies a match between the types *nat* and *num* (representing natural numbers).

rev_append	REV
∀ l, rev l = rev_append l [].	∀ L. REVERSE L = REV L []
∀ l l', rev_append l l' = rev l ++ l'.	∀ L1 L2. REV L1 L2 = REVERSE L1 ++ L2

Table 11. Properties shared by the Coq constant `rev_append` and the HOL4 constant `REV`.

rev_append	REV
Fixpoint rev_append (l l': list A) : list A :=	(∀ acc. REV [] acc = acc) ∧
match l with	∀ h t acc. REV (h::t) acc =
| [] => l'	REV t (h::acc)
| a::l => rev_append l (a::l')	
end.	

Table 12. Defintions of the Coq constant `rev_append` and the HOL4 constant `REV`.

All in all, the matching algorithm works across a variety of different theories (list, complex, probability, ...). This approach performs well on any kind of theories as long as developers of formal libraries state properties of the mathematical objects in a relatively similar manner. Still, distinguishing between an isomorphic construction and a structure sharing similar properties remains a challenge. We observe that running the algorithm in non-greedy mode enable us to obtain one-to-many mappings but those may also contain some non-optimal matches too. As an intermediate between the greedy method and the non-greedy strategy, a dynamic evaluation of the level of greediness (number of allowed counterparts of a constant) deepening the ideas used to implement the disambiguation option could be implemented.

5.5.3. Complexity and convergence

Applying a strategy does not come for free. The repetitive application of dynamical scoring is the most costly operation. To our advantage, the scoring updates takes less and less time to converge after each iteration. Indeed, the number of undecided matches diminishes. And the number of loops needed to reach a fixpoint is minimized since we restart the algorithm from the previous fixpoint. Let us take for example the process of aligning Coq and HOL4. The first scoring loop takes 7.37 seconds. But the greedy method with type coherence and disambiguation gives a total of 188 positive matches in 154 seconds.

This method of restarting from the previous fixpoint has experimentally always terminated. However, to guarantee convergence we would have to reset the scores after each update or only allow updates that do not mix positive and negative reinforcements. By construction, a positive (or negative) reinforcement will increase (respectively decrease) the current scores of all pairs. Therefore, updating scores after positive and negative reinforcement separately is enough to make the first step of the dynamical scoring monotonic. And this condition implies convergence as proved in Section 3.6.

6. Related Work

Our approach is in certain ways similar to latent semantic analysis (Landauer et al., 1998) used to find synonyms in multiple documents or text fragments. The relation is

Prover 1	Prover 2	Mode	Constant 1	Constant 2	rank
HOL4	HOL Light	G	*prod*	*prod*	1
			sum	*psum*	3
			RTC	*RTC*	269
		NG	(*prod real*) *real*	*complex*	251
HOL4	Isabelle/HOL	G	$\frac{\pi}{2}$	$\frac{\pi}{2}$	7
		NG	(*prod real*) *real*	*complex*	36
HOL Light	Isabelle/HOL	G	*real_pow*	*power real*	1
			ITLIST	*foldr*	95
Coq	Matita	G	*relation*	*relation*	1
			decidable	*decidable*	3
		NG	*Transitive N*	*transitive nat*	53
Coq	HOL4	G	*length*	*LENGTH*	3
			nat	*num*	4
			0_Z	*int_0*(*int_of_num* 0)	30
			Z	*int*	31
		NG	*BinNums_positive*	*num*	21
			BinNums_N	*num*	47
			BigN	*num*	48
Isabelle/HOL	Mizar	G	*pi*	*P_t*	7
			arccos	*arcos*	35
		NG	(*fold nat*) *nat***	$\longrightarrow$**	21
			member nat	*in*	3
Coq	Mizar	G	*PI*	*P_t*	90
			ALT_PI		
			Rlist	*FinSequence REAL*	106

Table 13. Interesting optimal matches found by running a strategy with disambiguation and type coherence in greedy mode (G) and non-greedy mode (NG). The presented non-greedy matches are not found by the greedy algorithm. The match (**) may not be an optimal one.

even more obvious if we consider our properties to be documents and concepts to be words. Similar pieces of text should contain words similar in meaning in the same way that similar properties contains similar concepts. However, our approach is able to use the structure of the properties, which cannot be done for informal documents.

A number of translations between formal mathematical libraries are able to use given matched concept. For this, usually matching concepts have been found manually. The first translation that introduced maps between concepts was the one of Obua and Skalberg (2006). There, two commands for mapping constructs were introduced: `type-maps` and `const-maps` which allow a user to map HOL Light and HOL4 concepts to corresponding ones in Isabelle/HOL. Given a type (or constant) in the maps, during the import of a theorem all occurrences of this type in the source system are replaced by the given type of the target system. In order for this construction to work, the basic properties of the concepts must already exist in the target system, and their translation must be avoided. Due to the complexity of finding such existing concepts and specifying the theorems which do not need to be translated, Obua and Skalberg were able to map only small

number of concepts like booleans and natural numbers, leaving integers or real numbers as future work.

The translation of Keller and Werner (2010) was the first one, which was able to map concepts between systems based on different foundations. The translation from HOL Light to Coq proceeds in two phases. First, the HOL proofs are imported as a defined structure. Second, using the *reflection* mechanism, native Coq properties are built. It is the second phase that allows mapping the HOL concepts like natural numbers to the Coq standard library type `N`.

The translation that maps so far the biggest number of concepts has been done by the second author (Kaliszyk and Krauss, 2013). The translation process consists of three phases, an exporting phase, offline processing and an import phase. The offline processing provides a verification of the (manually defined) set of maps and checks that all the needed theorems will be either skipped or mapped. This allows to quickly add mappings without the expensive step of performing the actual proof translation, and in turn allows for mapping 70 HOL Light concepts to their corresponding Isabelle/HOL counterparts. All these concept maps have been found and provided manually.

Bortin et al. (2006) implemented the AWE framework which allows the reuse of Isabelle/HOL formalization recorded as a proof trace multiple times for different concepts. Theory morphisms and parametrization are added to a theorem prover creating objects with similar properties. The use of theory morphisms together with concept mappings is one of the basic features of the MMT framework (Rabe, 2013). This allows for mapping concepts and theorems between theories also in different logics. So far all the mappings have been done completely manually.

Hurd's OpenTheory (Hurd, 2011) aims to share specifications and proofs between different HOL systems by defining small theory packages. In order to write and read such theory packages by theorem prover implementations a fixed set of concepts is defined that each prover can map to. This provides highest quality standard among the HOL systems, however since the procedure requires manual modifications to the sources and inspection of the libraries in order to find the mappings, so far only a small number of constants and types could be mapped. Similar aims are shared by semi-formal standardizations of mathematics, for example in the OpenMath content dictionaries. For a translation between semi-formal mathematical representation again concept lookup tables are constructed manually (So and Watt, 2006; Carlisle et al., 2001).

The Dedukti proof checker (Dowek et al., 2003), based on the $\lambda\Pi$-modulo calculus, can import and verify developments from Coq and HOL systems. An example Coq proof has been shown to be translatable to Dedukti and to instantiated with HOL natural numbers (Assaf and Cauderlier, 2015). One of the main challenges was to match the different typing levels of Coq and HOL into a common structure in the logic of Dedukti.

The proof advice systems for interactive theorem proving have studied similar concepts using various similarity measures. The methods have so far been mostly restricted to similarity of theorems and definitions. They have also been limited to single prover libraries. Heras and Komendantskaya in the proof pattern work Heras and Komendantskaya (2014) try to find similar Coq/SSReflect definitions using machine learning. Hashing of definitions in order to discover constants with same definitions in Flyspeck has been done in (Kaliszyk and Urban, 2014a). Searching for similar lemmas in order to find interesting properties has been tried for Mizar using the MoMM system (Urban, 2006a) as well as for HOL Light intermediate lemmas (Kaliszyk and Urban, 2015).

7. Conclusion

We have developed a methodology for matching concepts across formal mathematical libraries. Our approach relies on measuring the similarity of the properties of those concepts complemented by a dynamical process that iterates through their structural interrelation. Additional techniques such as subterm conceptualization and disambiguation appear to be highly beneficial to the quality of the matches and in some cases essential to the matching process.

Our experiments on multiple proof assistant libraries lead to the discovery of thousands of similar concept pairs. The full method performs particularly well between provers based on higher-order logic and variants of type theory. Aligning set-theoretical and type-theoretical provers automatically gives a smaller number of perfect matches.

8. Future works

We have focused on heuristic ways to match concepts avoiding the use of metadata, such as the names of the theories, theorems, and constants. Such metadata, as well as scoring heuristics refined by an unsupervised machine learning process could be used in practical applications of matches.

Furthermore, it would be interesting to test the quality of the found matches in the various applications. Sharing proof knowledge (Gauthier and Kaliszyk, 2015) could be performed across the studied libraries that have learning-assisted reasoning support (Blanchette et al., 2016). An early experiment with conjecturing (Gauthier et al., 2016) through analogies created from concept matches indicates some success. But more approaches (Kaliszyk et al., 2015) to transfer and create properties using knowledge from different mathematical domains could be tried. We would also like to provide a database of concept matches to create the possibility for external users to exploit the data for their own applications.

Acknowledgements

We thank Pierre Boutillier, Pierre-Marie Pédrot, Enrico Tassi and Yves Bertot for their help with creating a Coq plugin to export Coq formulas at the first "Coq coding sprint". We thank Josef Urban for his export of the Mizar library which we rely on. We valued the discussions with Dennis Müller, Florian Rabe, and Michael Kohlhase on the theoretical concept of theory morphism and their applications. We appreciated Chad Brown for his comments on the evaluation results. This research was supported by ERC starting grant no. 714034 *SMART*.

References

Asperti, A., Ricciotti, W., Coen, C. S., 2014. Matita tutorial. J. Formalized Reasoning 7 (2), 91–199.
URL http://dx.doi.org/10.6092/issn.1972-5787/4651

Aspinall, D., Kaliszyk, C., 2016. What's in a theorem name? (rough diamond). In: Blanchette, J., Merz, S. (Eds.), 7th Conference on Interactive Theorem Proving (ITP 2016). Vol. 9807 of LNCS. Springer, pp. 459–465.

Assaf, A., Cauderlier, R., 2015. Mixing HOL and Coq in Dedukti (extended abstract). In: Kaliszyk, C., Paskevich, A. (Eds.), Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015. Vol. 186 of EPTCS. pp. 89–96.

Attal, S., 2010. Markov chains and dynamical systems: The open system point of view. Communications on Stochastic Analysis 4, 523–540.

Awodey, S., 2006. Category theory, volume 49 of Oxford Logic Guides.

Bancerek, G., Byliński, C., Grabowski, A., Korniłowicz, A., Matuszewski, R., Naumowicz, A., Pąk, K., Urban, J., 2015. Mizar: State-of-the-art and beyond. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (Eds.), Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings. Vol. 9150 of LNCS. Springer, pp. 261–279.

Barbarossa, M., 2011. Stability of discrete dynamical systems. Matrix 21, a22.

Blanchette, J. C., Kaliszyk, C., Paulson, L. C., Urban, J., 2016. Hammering towards QED. J. Formalized Reasoning 9 (1), 101–148.
URL http://jfr.unibo.it/article/view/4593

Blanqui, F., 2003. Rewriting modulo in deduction modulo. In: Nieuwenhuis, R. (Ed.), Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings. Vol. 2706 of LNCS. Springer, pp. 395–409.

Bortin, M., Broch Johnsen, E., Lüth, C., 2006. Structured formal development in Isabelle. Nordic Journal of Computing 13, 1– 20.

Carlisle, D., Davenport, J., Dewar, M., Hur, N., Naylor, W., 2001. Conversion between MathML and OpenMath. Tech. Rep. 24.969, The OpenMath Society.

Corry, L., 2012. Modern algebra and the rise of mathematical structures. Birkhäuser.

de Moura, L. M., Kong, S., Avigad, J., van Doorn, F., von Raumer, J., 2015. The Lean theorem prover (system description). In: Felty, A. P., Middeldorp, A. (Eds.), Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings. Vol. 9195 of LNCS. Springer, pp. 378–388.

Dietrich, D., Whiteside, I., Aspinall, D., 2013. Polar: A framework for proof refactoring. In: McMillan, K. L., Middeldorp, A., Voronkov, A. (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings. Vol. 8312 of LNCS. Springer, pp. 776–791.

Dowek, G., Hardin, T., Kirchner, C., 2003. Theorem proving modulo. J. Autom. Reasoning 31 (1), 33–72.
URL http://dx.doi.org/10.1023/A:1027357912519

Gauthier, T., Kaliszyk, C., 2014. Matching concepts across HOL libraries. In: Watt, S., Davenport, J., Sexton, A., Sojka, P., Urban, J. (Eds.), Proc. of the 7th Conference on Intelligent Computer Mathematics (CICM'14). Vol. 8543 of LNCS. Springer Verlag, pp. 267–281.

Gauthier, T., Kaliszyk, C., 2015. Sharing HOL4 and HOL Light proof knowledge. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (Eds.), 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2015). Vol. 9450 of LNCS. Springer, pp. 372–386.

Gauthier, T., Kaliszyk, C., Urban, J., 2016. Initial experiments with statistical conjecturing over large formal corpora. In: Kohlhase, A., Libbrecht, P., Miller, B. R., Naumowicz, A., Neuper, W., Quaresma, P., Tompa, F. W., Suda, M. (Eds.), Joint Proceedings of the FM4M, MathUI, and ThEdu Workshops, Doctoral Program, and Work in Progress at the Conference on Intelligent Computer Mathematics 2016 (CICM-WiP 2016). Vol. 1785 of CEUR. CEUR-WS.org, pp. 219–228.

Graf, P., 1996. Term Indexing. Vol. 1053 of LNCS. Springer.
URL http://dx.doi.org/10.1007/3-540-61040-5

Haftmann, F., Krauss, A., Kunčar, O., Nipkow, T., 2013. Data refinement in Isabelle/HOL. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (Eds.), Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13). Vol. 7998 of LNCS. Springer, pp. 100–115.

Harrison, J., 2009. HOL Light: An overview. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (Eds.), TPHOLs. Vol. 5674 of LNCS. Springer, pp. 60–66.

Harrison, J., 2013. The HOL Light theory of Euclidean space. J. Autom. Reasoning 50 (2), 173–190.

Harrison, J., Urban, J., Wiedijk, F., 2014. History of interactive theorem proving. In: Siekmann, J. H. (Ed.), Computational Logic. Vol. 9 of Handbook of the History of Logic. Elsevier, pp. 135–214.

Hecht-Nielsen, R., 1989. Theory of the backpropagation neural network. In: Neural Networks, 1989. IJCNN., International Joint Conference on. IEEE, pp. 593–605.

Heras, J., Komendantskaya, E., 2014. Proof pattern search in Coq/SSReflect. arXiv preprint, CoRR abs/1402.0081.

Hirsch, M. W., 1988. Stability and convergence in strongly monotone dynamical systems. Journal für die reine und angewandte Mathematik 383, 1–53.
URL http://eudml.org/doc/152991

Huet, G., Herbelin, H., 2014. 30 years of research and development around Coq. In: Jagannathan, S., Sewell, P. (Eds.), The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014. ACM, pp. 249–250.

Hurd, J., 2011. The OpenTheory standard theory library. In: Bobaru, M. G., Havelund, K., Holzmann, G. J., Joshi, R. (Eds.), NASA Formal Methods. Vol. 6617 of LNCS. Springer, pp. 177–191.

Jacquel, M., Berkani, K., Delahaye, D., Dubois, C., 2015. Verifying B proof rules using deep embedding and automated theorem proving. Software and System Modeling 14 (1), 101–119.
URL http://dx.doi.org/10.1007/s10270-013-0322-z

Jones, K. S., 1972. A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation 28, 11–21.

Kaliszyk, C., Krauss, A., 2013. Scalable LCF-style proof translation. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (Eds.), Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13). Vol. 7998 of LNCS. Springer, pp. 51–66.
URL http://dx.doi.org/10.1007/978-3-642-39634-2_7

Kaliszyk, C., Mamane, L., Urban, J., 2014. Machine learning of Coq proof guidance: First experiments. In: Kutsia, T., Voronkov, A. (Eds.), Symbolic Computation in Software Science (SCSS 2014). Vol. 30 of EasyChair Proceedings in Computing. EasyChair, pp. 27–34.

Kaliszyk, C., Urban, J., 2014a. HOL(y)Hammer: Online ATP service for HOL Light, arXiv preprint abs/1309.4962, accepted for publication in Mathematics in Computer Science.

Kaliszyk, C., Urban, J., 2014b. Learning-assisted automated reasoning with Flyspeck. Journal of Automated Reasoning 53 (2), 173–213.

Kaliszyk, C., Urban, J., 2015. Learning-assisted theorem proving with millions of lemmas. Journal of Symbolic Computation 69, 109–128.

Kaliszyk, C., Urban, J., Vyskočil, J., 2015. Learning to parse on aligned corpora. In: Urban, C., Zhang, X. (Eds.), Interactive Theorem Proving (ITP 2015). Vol. 9236 of LNCS. pp. 227–233.

Keller, C., Werner, B., 2010. Importing HOL Light into Coq. In: Kaufmann, M., Paulson, L. C. (Eds.), ITP. Vol. 6172 of LNCS. Springer, pp. 307–322.

Klein, G., 2014. Proof engineering considered essential. In: Jones, C. B., Pihlajasaari, P., Sun, J. (Eds.), FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings. Vol. 8442 of LNCS. Springer, pp. 16–21.

Landauer, T. K., Foltz, P. W., Laham, D., 1998. An Introduction to Latent Semantic Analysis. Discourse Processes 25, 259–284.

Meyer, B., 1988. Object-oriented software construction. Vol. 2. Prentice hall New York.

Myreen, M. O., Davis, J., 2014. The reflective Milawa theorem prover is sound - (down to the machine code that runs it). In: Klein, G., Gamboa, R. (Eds.), Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings. Vol. 8558 of LNCS. Springer, pp. 421–436.
URL http://dx.doi.org/10.1007/978-3-319-08970-6

Naumowicz, A., Korniłowicz, A., 2009. A brief overview of Mizar. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (Eds.), Theorem Proving in Higher Order Logics. Vol. 5674 of LNCS. Springer Berlin Heidelberg, pp. 67–72.
URL http://dx.doi.org/10.1007/978-3-642-03359-9_5

Obua, S., Skalberg, S., 2006. Importing HOL into Isabelle/HOL. In: Furbach, U., Shankar, N. (Eds.), IJCAR. Vol. 4130 of LNCS. Springer, pp. 298–302.

Otten, J., 2005. Clausal connection-based theorem proving in intuitionistic first-order logic. In: Beckert, B. (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2005). Vol. 3702 of LNCS. Springer, pp. 245–261.

Paulson, L. C., 2016. Isabelle's logics: FOL and ZF.

Rabe, F., 2013. The MMT API: A generic MKM system. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (Eds.), Proc. of the 6th Conference on Intelligent Computer Mathematics (CICM'13). Vol. 7961 of LNCS. Springer, pp. 339–343.

Ramakrishnan, I. V., Sekar, R. C., Voronkov, A., 2001. Term indexing. In: Robinson, J. A., Voronkov, A. (Eds.), Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press, pp. 1853–1964.

Rotman, J. J., 2010. Advanced modern algebra. Vol. 114. American Mathematical Soc.

Schulz, S., 2013. Simple and efficient clause subsumption with feature vector indexing. In: Bonacina, M. P., Stickel, M. E. (Eds.), Automated Reasoning and Mathematics - Essays in Memory of William W. McCune. Vol. 7788 of LNCS. Springer, pp. 45–67.

Slind, K., Norrish, M., 2008. A brief overview of HOL4. In: Mohamed, O. A., Muñoz, C. A., Tahar, S. (Eds.), TPHOLs. Vol. 5170 of LNCS. Springer, pp. 28–32.

So, C. M., Watt, S. M., 2006. On the conversion between content MathML and OpenMath. In: Proc. of the Conference on the Communicating Mathematics in the Digital Era, (CMDE'06). pp. 169–182.

Sutcliffe, G., Schulz, S., Claessen, K., Baumgartner, P., 2012. The TPTP typed first-order form with arithmetic. In: Bjørner, N., Voronkov, A. (Eds.), LPAR. Vol. 7180 of LNCS. Springer, pp. 406–419.

Trybulec, A., 2007. Checker, compiled by Freek Wiedijk.
URL http://www.cs.ru.nl/F.Wiedijk/mizar/by.pdf

Univalent Foundations Program, T., 2013. Homotopy Type Theory: Univalent Foundations of Mathematics. https://homotopytypetheory.org/book, Institute for Advanced Study.

Urban, J., 2006a. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. Int. J. on Artificial Intelligence Tools 15 (1), 109–130.
URL http://ktiml.mff.cuni.cz/~urban/MoMM/momm.ps

Urban, J., 2006b. MPTP 0.2: Design, implementation, and initial experiments. J. Autom. Reasoning 37 (1-2), 21–43.
URL http://dx.doi.org/10.1007/s10817-006-9032-3

Walukiewicz, D., 1998. A total AC-compatible reduction ordering on higher-order terms. In: Larsen, K. G., Skyum, S., Winskel, G. (Eds.), Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings. Vol. 1443 of LNCS. Springer, pp. 530–542.

Wenzel, M., Paulson, L. C., Nipkow, T., 2008. The Isabelle framework. In: Mohamed, O. A., Muñoz, C. A., Tahar, S. (Eds.), Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings. Vol. 5170 of LNCS. pp. 33–38.

Whiteside, I., Aspinall, D., Dixon, L., Grov, G., 2011. Towards formal proof script refactoring. In: Davenport, J. H., Farmer, W. M., Urban, J., Rabe, F. (Eds.), Intelligent Computer Mathematics - 18th Symposium, Calculemus 2011, and 10th International Conference, MKM 2011, Bertinoro, Italy, July 18-23, 2011. Proceedings. Vol. 6824 of LNCS. Springer, pp. 260–275.