Matching concepts across HOL libraries

Thibault Gauthier and Cezary Kaliszyk

University of Innsbruck, Austria
{thibault.gauthier,cezary.kaliszyk}@uibk.ac.at

Abstract. Many proof assistant libraries contain formalizations of the same mathematical concepts. The concepts are often introduced (defined) in different ways, but the properties that they have, and are in turn formalized, are the same. For the basic concepts, like natural numbers, matching them between libraries is often straightforward, because of mathematical naming conventions. However, for more advanced concepts, finding similar formalizations in different libraries is a non-trivial task even for an expert.
In this paper we investigate automatic discovery of similar concepts across libraries of proof assistants. We propose an approach for normalizing properties of concepts in formal libraries and a number of similarity measures. We evaluate the approach on HOL based proof assistants HOL4, HOL Light and Isabelle/HOL, discovering 398 pairs of isomorphic constants and types.

1 Introduction

Large parts of mathematical knowledge formalized in various theorem provers correspond to the same informal concepts. Basic structures, like integers, are often formalized not only in different systems, but sometimes also multiple times in the same system. There are many possible reasons for this: the user may for example want to investigate special features available only for certain representations (like code extraction [4]), or simply check if the formal proofs can be done in a more straightforward manner with the help of alternate definitions. With multiple proof assistants, even the definitions of basic concepts may be significantly different: in Isabelle/HOL [21] the integers are defined as a quotient of pairs of naturals, while in HOL Light [6] they are a subset of the real numbers. Typically the proofs concerning a mathematical concept formalized in one system are not directly usable in the other, so a re-formalization is necessary.

The idea of exchanging formal developments between systems has been investigated both theoretically and practically many times [10, 14, 16]. Typically when a concept from the source systems is translated to a target system, and the same concept exists in the target system already, a new isomorphic structure is created and the relation between the two is lost. The properties that the two admit are the same and it is likely that the user formalized many similar ones.

In this work we investigate automatic discovery of such isomorphic structures mostly in the context of higher order logic. Specifically the contributions of this work are:

- We define patterns and properties of concepts in a formal library and export the data about constants and types from HOL Light, HOL4, and Isabelle/HOL together with the patterns.
- We investigate various scoring functions for automatic discovery of the same concepts in a library and across formal libraries and evaluate their performance.
- We find 398 maps between types and constants of the three libraries and show statistics about the same theorems in the libraries, together with normalization of the shape of theorems.

There exists a number of translations between formal libraries. The first translation of proofs that introduced maps between concepts was the one of Obua and Skalberg [16]. There, two commands for mapping constructs were introduced: `type-maps` and `const-maps` that let a user 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 first translation that mapped concepts of significantly different systems was the one of Keller and Werner [14]. The translation from HOL Light to Coq proceeds in two phases. First, the HOL proofs are imported as a defined structures. Second, thanks to 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 [10]. 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 such maps had to be provided manually.

Bortin et al. [1] 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 [17]. 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 [9] 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 [2, 19].

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 [8] 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 [12]. Using subsumption in order to find duplicate lemmas has been explored in the MoMM system [20] and applied to HOL Light lemmas in [11].

The rest of this paper is organized as follows: in Section 2 we describe the process of exporting the concepts like types and constants from three provers. In Section 3 we discuss the classification of patterns together with the normalization of theorems, while in Section 4 we define the scoring functions and an iterative matching algorithm.We present the results of the experiments in Section 5 and in Section 6 we conclude and present an outlook on the future work.

2 The theorem and constant data

In this section we shortly describe the data that we will perform our experiments on and the way the theorems and constants are normalized and exported. We chose three proof assistants based on higher-order logic: HOL4 [18], HOL Light [6] and Isabelle/HOL [21]. The sizes of the core libraries of the three are significantly different, so in order to get more meaningful results we export library parts of the same order of magnitude. This amounts to all the theories included with the standard distribution of HOL4. In case of HOL Light we include multivariate analysis [7], HOL in HOL [5] and the 67 files that include the proofs of the 100 theorems [22] compatible with the two. For Isabelle we export the theory `Main`.

The way to access all the theorems and constants in HOL Light has been described in detail in [13] and for HOL4 and Isabelle/HOL accessing values of theories can be performed using the modules provided by the provers (`DB.thms` and `@{theory}` object respectively). We first perform a minimal normalization of the forms of theorems (a further normalization will be performed on the common representation in Section 3) and export the data. We will focus on HOL4, the procedures in the other two are similar.

The hypotheses of the theorems are discharged and all free variables are generalized. In order to avoid patterns arising from known equal constants, all theorems of the form $\vdash c_1 = c_2$ (in HOL4 four of them are found by calling `DB.match`) are used to substitute c_1 by c_2 in all theorems.

The named theorems and constants are prefixed with theory names and explicit category classifiers (`c` for constants, `t` for theorems) to avoid ambiguities. Similarly, variables are explicitly numbered with their position of the binding λ (this is equivalent to the de Bruijn notation, but possible within the data structure used by each of the three implementations). We decided to include the type information only at the constant level, and to skip it inside the formulas.

Example 1. $\forall x : int.\ x = x \longrightarrow cHOL4.bool.\forall\ (\lambda V.((cHOL4.min.=\ \ V)\ V))$

Analogously, for all the constants their most general types are exported. Type variables are normalized using numbers that describe their position and type constructors are prefixed using theory identifiers and an explicit type constructor classifier.

Example 2. $(num, a) \longrightarrow tHOL4.pair.prod(tHOL4.num.num, Aa)$

The numbers of exported theorems and constants are presented in Table 1.

	HOL Light	HOL4	Isabelle/HOL
Number of theorems	11501	10847	18914
Number of constants	871	1962	2214

Table 1: Number of theorems and constants after the exporting phase

3 Patterns and classification

In this section we will look at the concept of *patterns* created from theorems, which is crucial in our classification of concepts and the algorithms for deriving patterns and matching them. In the following we will call the constants and types already mapped to concepts as *defined*.

Definition 1 (pattern). *Let f be a formula with no free variables and C the set of its constants. Let $D = \{d_1, \ldots, d_n\}$ be a set of defined constants and $A = C \setminus D = \{a_1, \ldots, a_m\}$ a set of undefined constants. Its pattern is defined by:*

$$P(f[a_1, \ldots, a_m, d_1, \ldots, d_n]) := \lambda a_1 \ldots a_n.f[a_1, \ldots, a_n, d_1, \ldots, d_n]$$

Example 3. The pattern of $\forall x\ y.\ x * y = y * x$ is:
- with $D = \{\forall, =\}$, $\quad \lambda a_1.\ \forall x\ y.\ a_1\ x\ y = a_1\ y\ x$.
- with $D = \{\forall\}$, $\quad \lambda a_1 a_2.\ \forall x\ y.\ a_1(a_2\ x\ y)(a_2\ y\ x)$.
- with $D = \varnothing$, $\quad \lambda a_1 a_2 a_3.\ a_1\ \lambda x\ y.\ a_2(a_3\ x\ y)(a_3\ y\ x)$.

Patterns are equal when they are α-equivalent. In practice, we order the variables and constants by the order in which they appear when traversing the formula from top to bottom. This means that checking if two formulas are α-equivalent amounts to verifying the equality of their patterns with no constants abstracted.

The formulas exported from all proof assistant libraries are parsed to a standard representation (λ-terms). The basic logical operators of the different provers are mapped to the set of defined constants and the theorems are rewritten using these mappings before further normalization. Finally, the patterns of the normalized formulas are extracted according to the specified defined constants.

We define three ways in which patterns are derived from the formula, each corresponding to a certain level of normalization:

$norm_0$: Given $D = \varnothing$ we can define a pattern corresponding to the theorem without any abstraction (identity).

$norm_1$: With $D = \{\forall, \exists, \wedge, \vee, \Rightarrow, \neg, =\}$ ($\Leftrightarrow$ is considered as $=$). The procedure is similar to the normalization done by first order provers (to the conjunctive normal form) with the omission of transformations on existential quantifiers, as we do not want do perform skolemization. We additionally normalize associative and commutative operations. The procedure performs the following steps at every formula level:

- remove implication,
- move negation in,
- move universal quantifiers out (existential quantifiers are not moved out to maximize the number of disjunctions in the last step),
- distribute disjunction over conjunctions,
- rewrite based on the associativity of $\forall, \exists, \wedge$ and $\vee$,
- rewrite based on the commutativity of $\forall, \exists, \wedge, \vee$ and $=$,
- separate disjunctions at the top formula level (example below).

Example 4. $\forall x\ y.\ (x \geq 0 \wedge x \leq y) \longrightarrow (\forall x.\ x \geq 0) \wedge (\forall x\ y.\ x \leq y)$

$norm_2$: Aside from all the normalizations performed by $norm_1$, we additionally consider a given list of associative and commutative constants (see Table 2 in Section 5) that is used to further normalize the formula. The set of defined constants stays the same as $norm_1$, which in particular means that the associative - commutative (AC) constants stay undefined and can be abstracted over.

Given the normalized theorems we will look at patterns relative to constants. In the following, we will assume that the constants are partitioned in ones that have been defined (mapped to a constant) and undefined.

Definition 2 (pattern relative to a constant). *Let a_{i-1} be an undefined constant appearing in a formula f in the i^{th} position. The pattern of f relative to a_{i-1} is defined by:*

$$P_{a_{i-1}}(f) := (P(f), i-1)$$

Example 5. Suppose $D = \varnothing$. Then the only two patterns that the reflexivity principle induces are:

$$P_{\forall}(\forall x.\ x = x) = (\lambda a_0 a_1.\ a_0\ (\lambda v_0.\ a_1\ v_0\ v_0), 0)$$
$$P_{=}(\forall x.\ x = x) = (\lambda a_0 a_1.\ a_0\ (\lambda v_0.\ a_1\ v_0\ v_0), 1)$$

Typically, we will be interested in patterns where D includes the predicate logic constants, so the reflexivity principle will not produce any patterns. The patterns will be properties of operations like commutativity or associativity. In order to find all such properties we define:

Definition 3. *The set of patterns associated with a constant c in a library lib is defined by:*

$$P^{set}(lib, c) = \bigcup_{f \in lib} P_c(f)$$

Let (abs, i) be a relative pattern. Its associated set of constants, in library lib, is:

$$C^{set}(lib, (abs, i)) := \{c \in lib, \exists f \in lib,\ \ P_c(f) = (abs, i))\}$$

We can now define one of the basic measures we will use for comparing similarity of constants:

Definition 4. *The set of common relative patterns shared by a constant c_1 in lib_1, and a constant c_2 in lib_2 is:*

$$P^{set}(lib_1, c_1) \cap P^{set}(lib_2, c_2)$$

In the remaining part of this paper, we will not always specify if a pattern is relative or not.

We proceed with forming type patterns. Type patterns are defined in a similar way to formula patterns. Types are partitioned into already defined types (initially the type of booleans – propositions) and undefined types. Type variables are also considered as undefined to enable their instantiation, and the list of leaf and node types involved is saved to allow matching.

Example 6. Let $D^{type} = \{fun\}$ and a be a type variable. Then:

$$\begin{aligned} P^{type}((a \to a, int \to int)) &= P^{type}((pair(fun(a, a), fun(int, int)))) \\ &= (\lambda a_0 a_1 a_2.\ (a_0(fun(a_1, a_1), fun(a_2, a_2))), [pair, a, int]) \end{aligned}$$

Suppose we are given two types with respective patterns $(abs_1, [t_1 \dots t_n])$ and $(abs_2, [u_1 \dots u_m])$. They match if abs_1 is α-equivalent to abs_2. The list of their derived type matches is $[(t_1, u_1), \dots, (t_n, u_n)]$, from which the pairs containing at least one type variable are removed.

4 Matching concepts across libraries

In this section, we will investigate measures of similarity in order to find the same types and constants between libraries. First, we will define a similarity score for each pair of constants. Then, we will suppose that the best match is correct and use it to update the similarity scores of the other pairs iteratively.

4.1 Similarity score

The easiest way to tell if two constants are related is to look at the number of patterns they share. However, the more a pattern has associated constants, the less relevant it is. To test each of these possibilities, two weighting functions are defined:

$$w_0(lib,p) = 1, \quad w_1(lib,p) = \frac{1}{card(C^{set}(lib,p))}$$

where p is a pattern in library lib. The weighting functions presented here do not consider the size of the pattern, nor the numbers of defined and undefined constants. Considering more complicated weighting functions may be necessary for formal libraries with significantly different logics.

Based on the weighting functions two scoring functions are defined. Let c_1 be a constant from library lib_1 and c_2 a constant from library lib_2. Let $P = \{p_1, \ldots, p_k\}$ be the set of patterns c_1 and c_2 have in common. Then:

$$score_0(c1,c2) = \sum_{i=1}^{k} w_0(lib_1,p_i) * w_0(lib_2,p_i)$$

$$score_1(c1,c2) = \sum_{i=1}^{k} w_1(lib_1,p_i) * w_1(lib_2,p_i)$$

In order to account for the fact that constants with a high number of associated patterns are more likely to have common patterns with unrelated constants, we further modify $score_1$. Let n_1 be the number of patterns associated to c_1 and n_2 be the number of patterns associated to c_2. We define a third similarity scoring function by:

$$score_2(c1,c2) = \frac{\sum_{i=1}^{k} w_1(lib_1,p_i) * w_1(lib_2,p_i)}{log(2 + n_1 * n_2)}$$

4.2 Iterative approach

In our initial experiments, a direct computation of the $score_i$ functions for all constants in two libraries after an initial number of correct pairs would find incorrect pairs (false positive matches). Such pairs can be quickly eliminated if the information coming from the first successful matches is propagated further.

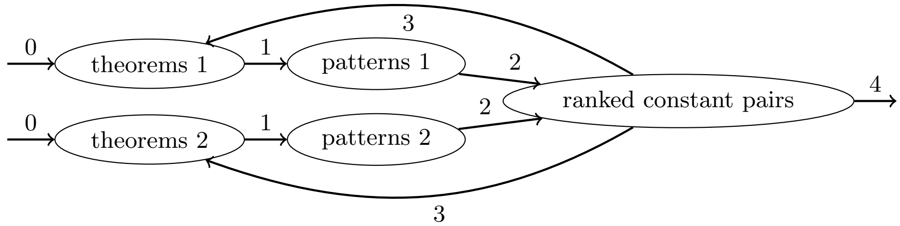

Fig. 1. Graphical representation of the iterative procedure

In order to do this, we propose an iterative approach (presented schematically in Fig. 1):

The iterative approach returns a sorted list of pairs of constants and a sorted list of pair of types from two libraries by following this steps:

0. Export theorems from a library as well as constants with their types and parse them.
1. Normalize theorems and create theorem patterns, constant patterns and type patterns according to the current defined constants and types.
2. Score every pair of constants.
3. Take the highest ranked pair of constants (c_1, c_2). Check if their type matches, if not take the next one and so on. When their type matches, rewrite all the theorems inside lib_1 with the substitution $c_1 \rightarrow d$ and all the theorems inside lib_2 with the substitution $c_2 \rightarrow d$, where d is a fresh defined constant. Then, get the derived pairs of types from the pair of constant and substitute every pair member with the same fresh defined type as for the other member.
4. Return the pairs of constants and the pairs of types, in the order they were created, when you have reached the number of iteration desired.

The single-pass approach is defined by doing only one iteration, where the list of pairs of constants are returned ranked by their score. A type check performed after a single-pass can discard a number of wrong matches efficiently.

In the presented approach, we assume that the constants and types inside one library are all different, which we tried to ensure by the initial normalization. Thus, we will not match constant from the same library. Furthermore, if a constant is matched, then it can no longer be matched again and the same reasoning applies for types. This first statement will turn out not to be true for a few constants in Section 5.

The complexity of the iterative approach is obviously larger than that of the single-pass approach. On an IntelM 2.66GHz CPU, the single-pass approach between HOL4 and HOL Light with $score_2$ and $norm_2$ takes 6 minutes to complete. The main reason is that it has to compare the patterns of all possible pairs of constants (about two million). Thus, the bottleneck is the time taken by the comparison function which intersects the set of patterns associated with each constant and scores the resulting set. However, the iterative method can use

HOL Light			HOL4			Isabelle/HOL		
Pattern	Consts	Thms	Pattern	Consts	Thms	Pattern	Consts	Thms
Inj	37	37	Inj	54	68	Inj	83	137
Asso	32	36	Asso	50	65	App	17	18
Comm	25	44	Comm	40	48	Inj1	16	16
Refl	22	22	Trans	32	33	Comm	14	51
Lcomm	19	20	Refl	23	23	Inj2	12	35
Idempo	12	12	Idempo	20	15	App2	11	12

Table 2: Most frequent properties of one constant

the first pass to remove pairs of constants that have no common patterns. This reduces the number of possible matches to ten thousand. As a consequence, it takes only 3 minutes more to do 100 iterations.

5 Experiments

In order to verify the correctness of our approach we first investigate the most common patterns and shapes of theorems in each of the three formal libraries and then we look at the results of the matching constants across libraries. The data given by these experiments is available at `http://cl-informatik.uibk.ac.at/users/tgauthier/matching/`.

5.1 Single library results

Tables 2 and 3 show the most common properties when applying the standard normalization $norm_1$ of a single constant and of two constants respectively in the three considered proof assistant libraries. The tables are sorted with respect to the total number of different constants in the theorems from which the patterns are derived. In Table 2, *Inj* stands for injectivity, *Asso* for associativity and *Comm* for commutativity. In Table 3, the pattern *Class* and *Inv* are defined by $Class\ (c_0, c_1) = c_0\ c_1$, $Inv(c_0, c_1) = \forall x_0.\ c_0\ (c_1\ x_0) = x_0$.

As expected, HOL Light and HOL4 show the most similar results and injectivity is the most frequent property. Commutativity and associativity are also very common, and their associated constants are used to apply $norm_2$ as stated in Section 3.

The common patterns immediately show constants defined to be equivalent to the defined equality in each of the libraries, through an extensional definition. There is one such constant in HOL4, one in HOL Light and three in Isabelle/HOL. In order to avoid missing or duplicate patterns we mapped all these constants to equality manually.

HOL Light			HOL4			Isabelle/HOL		
Pattern	Consts	Thms	Pattern	Consts	Thms	Pattern	Consts	Thms
Class	71	87	Inv	131	89	Class	188	642
Inv	64	34	Neutr	64	55	Inv	114	75
Imp	52	76	Class	63	70	Equal	58	40

Table 3: Most frequent properties of two constants

Furthermore, in Table 3, the third row of the Isabelle/HOL column shows 40 equalities between two different constants that were created during the normalization. We have also found 10 such equalities in HOL4 and 1 in HOL Light. Often a constant with a less general type can be replaced by the other, but without type-class information in Isabelle/HOL we decided not to do such replacements in general.

5.2 Cross-library results

The way we analyze the quality of the matching, is by looking at the number of correct matches of types and constants between the libraries, in particular we consider the occurrence of the first incorrect match, also called *false positive* below. It is very hard to spot same concepts in two large libraries, therefore a manual evaluation of the false negatives (pairs that could be mapped but are not) is a very hard task and requires the knowledge of the whole libraries.

In the first three experiments, we test how much normalization, scoring, iteration and types contribute to better matches. This will be used to choose the best parameters for matching constants and types between each pair of provers.

The first experiment (Fig. 2) evaluates the similarity of the libraries. We match the provers using the (a-priori) strongest normalization ($norm_2$) with a single-pass approach with no types. In this setting, the constant with the most similar properties is 0 between HOL Light and HOL4, and between HOL4 and Isabelle/HOL. And it is ∅ between HOL Light and Isabelle/HOL. Form this perspective, the most similar pairs of provers are in decreasing order (HOL Light,HOL4), (HOL4,Isabelle/HOL) and (HOL Light-Isabelle/HOL). We test the four other parameters relative to the pair of provers (HOL Light, HOL4) as we should have most common patterns to work with.

The second experiment (Fig. 3) is meant to evaluate the efficiency of normalization on the number of patterns. It is also run as a single-pass with no types. We observe an increase in number of patterns from $norm_0$ and $norm_1$ which is mostly due to the splitting of disjunctions. Moreover, the difference between $norm_2$ and $norm_1$ is negligible, which means that associative and commutative constants are used in almost the same way across the two libraries. In the following experiments we will only use $norm_2$ assuming it is the strongest normalization also in the other scenarios.

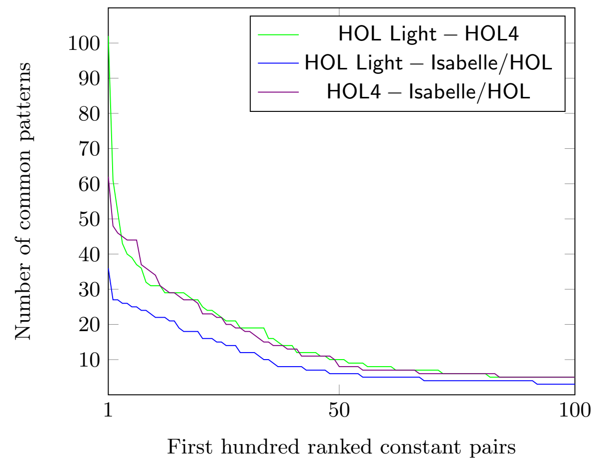

Fig. 2. Number of patterns by constant pairs in different provers

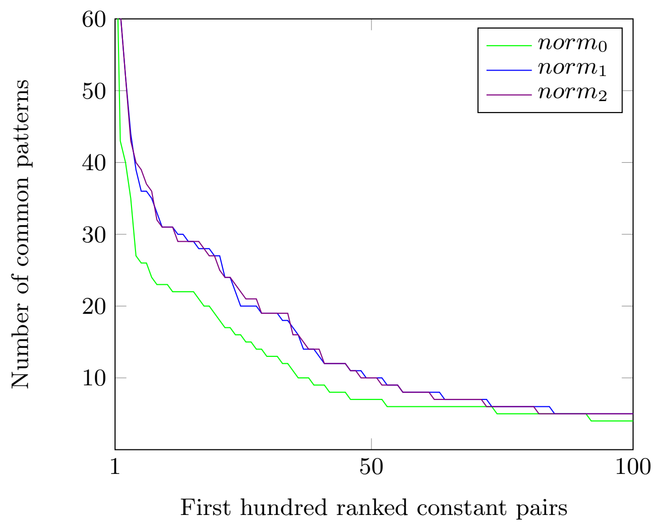

Fig. 3. The normalization effect

We next evaluate the scoring functions, the contribution of iterations, and of the use of type information. Table 4 shows the effect of iterative method and scoring function on the occurrence of the first wrong match (false positive). It has been inspected manually. Fig. 4, shows the positive effects of the iterative effect on the $score_1$ and $score_2$ curves. Some patterns are ranked higher after an iteration, as they become more scarce. The iterative method also has an opposite effect that is not directly visible in the figure: the score of pairs of constants diminishes by removing false pattern matches. Table 5 shows how type information contributes to matches. Types do help, but become less important with better scoring functions combined with the iterative approach.

	$score_0$	$score_1$	$score_2$
Single-pass	39	69	88
Iterative	49	68	113

Table 4: Rank of the first wrong match for (HOL Light, HOL4)

	$score_0$	$score_1$	$score_2$
Single-pass	31	19	21
Iterative	224	18	6

Table 5: Number of pairs of constants discarded, due to type matching

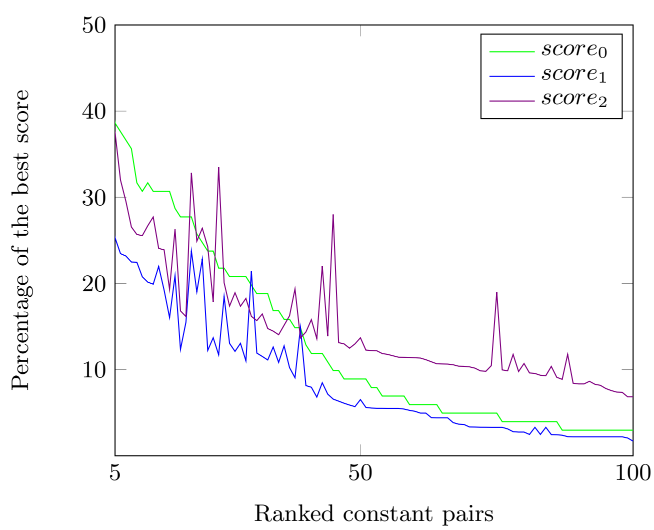

Fig. 4. Effect of different scoring functions on the iterative approach

The last experiment is run with the best parameters found by the previous experiments, namely $norm_2$, $score_2$ and the iterative approach with types. Three numbers are presented in each cell of Tables 6 and 7. The first one is the number of correct matches obtained before the first error. The second one is number of correct matches we have found. In the case of constants, the third one is the number of matches we have manually checked. We stop at a point where a previously found error propagates. In the case of types, the third number is the rank of the last correct match. As seen previously, the best results come from comparing the HOL4 and HOL Light libraries, where we have verified 177 constant matches and 16 type matches.

HOL Light-HOL4		HOL4-Isabelle/HOL		HOL Light-Isabelle/HOL	
112	177/203	65	109/131	55	78/98

Table 6: Number of constants accurately matched

HOL Light-HOL4		HOL4-Isabelle/HOL		HOL Light-Isabelle/HOL	
11	16/22	8	11/17	6	7/13

Table 7: Number of types accurately matched

6 Conclusion

We have investigated the formal mathematical libraries of HOL Light, HOL4 and Isabelle/HOL searching for common types and constants. We defined a concept of patterns that capture abstract properties of constants and types and normalization on theorems that allow for efficient computation of such patterns. The practical evaluation of the approach on the libraries let us find hundreds of pairs of common patterns, with a high accuracy.

Formal mathematical libraries contain many instances of the same algebraic structures. Such instances have many same properties therefore their matching is non-trivial. Our proposed approach can match such instances correctly, because of patterns that link such concepts to other concepts. For example integers and matrices are instances of the algebraic structure ring. However each of the libraries we analyzed contains a theorem that states that each integers is equal to a natural number or its negation. A pattern derived from this fact, together with many other patterns that are unique to integers match them across libraries correctly.

The work gives many correct matches between concepts that can be directly used in translations between proof assistants. In particular HOL/IMPORT would immediately benefit from mapping the HOL Light types and constants to their Isabelle/HOL counterparts allowing for further merging of the results.

The approach has been tested on three provers based on higher-order logic. In principle the properties of the standard mathematical concepts defined in many other logics are the same, however it remains to be seen how smoothly does the approach extend to provers based on different logics.

In order to further decrease the number of false positive matches, more weighting and scoring functions could be considered. One could imagine functions that take into account the length of formulas, and numbers of mapped concepts per pattern. Similarly, the scoring functions could penalize pairs of constants with only one pattern in common (these have been the first false positives in all our experiments). Further, the equalities between constants created during normalization could be used for further rewriting of theorems into normal forms. Other ideas include normalizing relatively to distributive constants and trying weaker kind of matching for example on atoms or subterms.

Building a set of basic mathematical concepts together with their foundational properties has been on the MKM wish-list for a long time. It remains to be seen if a set of common concepts across proof assistant libraries can be extended by minimal required properties to automatically build such "interface theories", and if automatically found larger sets of theories can complement the high-quality interface theories built in the MKM community.

Acknowledgments

We would like to thank Josef Urban for his comments on the previous version of this paper.
This work has been supported by the Austrian Science Fund (FWF): P26201.

References

1. Maksym Bortin, Einar Broch Johnsen, and Christoph Lüth. Structured formal development in Isabelle. *Nordic Journal of Computing*, 13:1– 20, 2006.
2. David Carlisle, James Davenport, Mike Dewar, Namhyun Hur, and William Naylor. Conversion between MathML and OpenMath. Technical Report 24.969, The OpenMath Society, 2001.
3. Ulrich Furbach and Natarajan Shankar, editors. *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.
4. Florian Haftmann, Alexander Krauss, Ondrej Kuncar, and Tobias Nipkow. Data refinement in Isabelle/HOL. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of *LNCS*, pages 100–115. Springer, 2013.
5. John Harrison. Towards self-verification of HOL Light. In Furbach and Shankar [3], pages 177–191.

6. John Harrison. HOL Light: An overview. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLs*, volume 5674 of *Lecture Notes in Computer Science*, pages 60–66. Springer, 2009.
7. John Harrison. The HOL Light theory of euclidean space. *J. Autom. Reasoning*, 50(2):173–190, 2013.
8. Jónathan Heras and Ekaterina Komendantskaya. Proof pattern search in Coq/SSReflect. *arXiv preprint, CoRR*, abs/1402.0081, 2014.
9. Joe Hurd. The OpenTheory standard theory library. In Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, volume 6617 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2011.
10. Cezary Kaliszyk and Alexander Krauss. Scalable LCF-style proof translation. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of *LNCS*, pages 51–66. Springer, 2013.
11. Cezary Kaliszyk and Josef Urban. Lemma mining over HOL Light. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'13)*, volume 8312 of *LNCS*, pages 503–517. Springer Verlag, 2013.
12. Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. arXiv preprint abs/1309.4962, accepted for publication in Mathematics in Computer Science, 2014.
13. Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. arXiv preprint abs/1211.7012, accepted for publication in Journal of Automated Reasoning, 2014.
14. Chantal Keller and Benjamin Werner. Importing HOL Light into Coq. In Matt Kaufmann and Lawrence C. Paulson, editors, *ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 2010.
15. Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors. *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *Lecture Notes in Computer Science*. Springer, 2008.
16. Steven Obua and Sebastian Skalberg. Importing HOL into Isabelle/HOL. In Furbach and Shankar [3], pages 298–302.
17. Florian Rabe. The MMT API: A generic MKM system. In Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors, *Proc. of the 6th Conference on Intelligent Computer Mathematics (CICM'13)*, volume 7961 of *LNCS*, pages 339–343. Springer, 2013.
18. Konrad Slind and Michael Norrish. A brief overview of HOL4. In Mohamed et al. [15], pages 28–32.
19. Clare M. So and Stephen M. Watt. On the conversion between content MathML and OpenMath. In *Proc. of the Conference on the Communicating Mathematics in the Digital Era, (CMDE'06)*, pages 169–182, 2006.
20. Josef Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. *Int. J. on Artificial Intelligence Tools*, 15(1):109–130, 2006.
21. Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. The Isabelle framework. In Mohamed et al. [15], pages 33–38.
22. Freek Wiedijk, editor. *The Seventeen Provers of the World*, volume 3600 of *Lecture Notes in Computer Science*. Springer, 2006.