Automatically Finding Theory Morphisms for Knowledge Management

Dennis Müller[1], Michael Kohlhase[1], and Florian Rabe[1,2][0000-0003-3040-3655]

[1] Computer Science, FAU Erlangen-Nürnberg
[2] LRI, Université Paris Sud

Abstract. We present a method for finding morphisms between formal theories, both within as well as across libraries based on different logical foundations. As they induce new theorems in the target theory for any of the source theory, theory morphisms are high-value elements of a modular formal library. Usually, theory morphisms are manually encoded, but this practice requires authors who are familiar with source and target theories at the same time, which limits the scalability of the manual approach.
To remedy this problem, we have developed a morphism finder algorithm that automates theory morphism discovery. In this paper we present an implementation in the MMT system and show specific use cases. We focus on an application of *theory discovery*, where a user can check whether a (part of a) formal theory already exists in some library, potentially avoiding duplication of work or suggesting an opportunity for refactoring.

1 Introduction

Motivation "Semantic Search" – a very suggestive term, which is alas seriously under-defined – has often been touted as the "killer application" of semantic technologies. With a view finder, we can add another possible interpretation: searching mathematical ontologies (here modular theorem prover libraries) at the level of theories – we call this **theory classification**. theoryclassification

The basic use case is the following: Jane, a mathematician, becomes interested in a class of mathematical objects, say – as a didactic example – something she initially calls "beautiful subsets" of a base set $\mathcal{B}$ (or just "beautiful over $\mathcal{B}$"). These have the following properties Q:

1. the empty set is beautiful over $\mathcal{B}$
2. every subset of a beautiful set is beautiful over $\mathcal{B}$
3. If A and B are beautiful over $\mathcal{B}$ and A has more elements than B, then there is an $x \in A \backslash B$, such that $B \cup \{x\}$ is beautiful over $\mathcal{B}$.

To see what is known about beautiful subsets, she types these three conditions into a theory classifier, which computes any theories in a library $\mathcal{L}$ that match these (after a suitable renaming). In our case, Jane learns that her "beautiful sets" correspond to the well-known structure of matroids [MWP], so she can directly apply matroid theory to her problems.

In extended use cases, a theory classifier find theories that share significant structure with Q, so that Jane can formalize Q modularly with minimal effort. Say Jane was interested in "dazzling subsets", i.e. beautiful subsets that obey a fourth condition, then she could just contribute a theory that extends matroid by a formalization of the fourth condition – and maybe rethink the name.

In this paper we reduce the theory classification problem to the problem of finding theory morphisms (views) between theories in a library $\mathcal{L}$: given a query theory Q, the algorithm computes all (total) views from Q into $\mathcal{L}$ and returns presentations of target theories and the assignments made by the views.

Related Work Existing systems have so far only worked with explicitly given views, e.g., in IMPS [FGT93] or Isabelle [Pau94]. Automatically and systematically searching for new views was first undertaken in [NK07] in 2006. However, at that time no large corpora of formalized mathematics were available in standardized formats that would have allowed easily testing the ideas in practice.

This situation has changed since then as multiple such exports have become available. In particular, we have developed the MMT language [RK13] and the concrete syntax of the OMDoc XML format [Koh06] as a uniform representation language for such corpora. And we have translated multiple proof assistant libraries into this format, among others those of PVS in [Koh+17]. Building on these developments, we are now able, for the first time, to apply generic methods — i.e., methods that work at the MMT level — to search for views in these libraries.

While inspired by the ideas of [NK07], our design and implementation are completely novel. In particular, the theory makes use of the rigorous language-independent definitions of *theory* and *view* provided by MMT, and the practical implementation makes use of the MMT system, which provides high-level APIs for these concepts.

[GK14] applies techniques related to ours to a related problem. Instead of views inside a single corpus, they use machine learning to find similar constants in two different corpora. Their results can roughly be seen as a single partial view from one corpus to the other.

Approach and Contribution Our contribution is twofold. Firstly, we present the design and implementation of a generic view finder that works with arbitrary corpora represented in MMT. The algorithm tries to match two symbols by unifying their types. This is made efficient by separating the term into a hashed representation of its abstract syntax tree (which serves as a fast plausibility check for pre-selecting matching candidates) and the list of symbol occurrences in the term, into which the algorithm recurses.

Secondly, we apply this view finder in two case studies: In the first, we start with an abstract theory and try to figure out if it already exists *in the same library* – the use case mention above. In the second example, we write down a simple theory of commutative operators in one language to find all commutative operators in *another library based on a different foundation.*

Overview In Section 2, we revise the basics of MMT and views. Section 3 presents the view finding algorithm restricted to the intra-library case and showcases it for the theory classification use case. In Section 4, we extend the algorithm to inter-library view finding discuss results of applying it to the PVS/NASA library. Section 5 concludes the paper and discusses additional applications.

A more extensive version of this paper with additional details can be found at [MRK].

2 Preliminaries: MMT and Views

Intuitively, MMT is a declarative language for theories and views over an arbitrary object language. Its treatment of object languages is abstract enough to subsume most logics and type theories which are practically relevant.

Fig. 1 gives an overview of the fundamental MMT concepts. In the simplest case, **theories** Σ are lists of **constant declarations** $c : E$, where E is an expression that may use the previously declared constants. Naturally, E must be subject to some type system (which MMT is also parametric in), but the details of this are not critical for our purposes here. We say that Σ' includes Σ if it contains every constant declaration of Σ.

meta-theory: a fixed theory M		
	Theory Σ	View $\sigma : \Sigma \to \Sigma'$
set of Σ-expressions E	typed constant declarations $c : E$ formed from M- and Σ-constants	assignments $c \mapsto E'$ mapped to Σ' expressions

Fig. 1. Overview of MMT Concepts

Correspondingly, a **view** $\sigma : \Sigma \to \Sigma'$ is a list of **assignments** $c \mapsto e'$ of Σ'-expressions e' to Σ-constants c. To be well-typed, σ must preserve typing, i.e., we must have $\vdash_{\Sigma'} e' : \overline{\sigma}(E)$. Here $\overline{\sigma}$ is the homomorphic extension of σ, i.e., the map of Σ-expressions to Σ'-expressions that substitutes every occurrence of a Σ'-constant with the Σ'-expression assigned by σ. We call σ **simple** if the expressions e' are always Σ'-*constants* rather than complex expressions. The type-preservation condition for an assignment $c \mapsto c'$ reduces to $\overline{\sigma}(E) = E'$ where E and E' are the types of c and c'. We call σ **partial** if it does not contain an assignment for every Σ-constant and **total** otherwise. A partial view from Σ to Σ' is the same as a total view from some theory included by Σ to Σ'.

Importantly, we can then show generally at the MMT-level that if σ is well-typed, then $\overline{\sigma}$ preserves all typing and equality judgments over Σ. In particular, if we represent proofs as typed terms, views preserve the theoremhood of propositions. This property makes views so valuable for structuring, refactoring, and integrating large corpora.

MMT achieves language-independence through the use of **meta-theories**: every MMT-theory may designate a previously defined theory as its meta-theory.

For example, when we represent the HOL Light library in MMT, we first write a theory L for the logical primitives of HOL Light. Then each theory in the HOL Light library is represented as a theory with L as its meta-theory. In fact, we usually go one step further: L itself is a theory, whose meta-theory is a logical framework such as LF. That allows L to concisely define the syntax and inference system of HOL Light.

However, for our purposes, it suffices to say that the meta-theory is some fixed theory relative to which all concepts are defined. Thus, we assume that Σ and Σ' have the same meta-theory M, and that $\overline{\sigma}$ maps all M-constants to themselves.

It remains to define the exact syntax of expressions. In the grammar on the right c refers to constants (of the meta-theory or previously declared in the current theory) and x refers to bound variables. Complex expressions are of the form $o\,[x_1 : t_1, \ldots, x_m : t_m]\,(a_1, \ldots, a_n)$, where

$$\begin{array}{lcl} \Gamma & ::= & (x : E)^* \\ E & ::= & c \mid x \mid E\,[\Gamma]\,(E^+) \end{array}$$

- o is the operator that forms the complex expression,
- $x_i : t_i$ declares variable of type t_i that are bound by o in subsequent variable declarations and in the arguments,
- a_i is an argument of o

The bound variable context may be empty, and we write $o\,(\vec{a})$ instead of $o\,[\cdot]\,(\vec{a})$. For example, the axiom $\forall x : \texttt{set}, y : \texttt{set}.\ \texttt{beautiful}(x) \wedge y \subseteq x \Rightarrow \texttt{beautiful}(y)$ would instead be written as

$$\forall\,[x : \texttt{set}, y : \texttt{set}]\,(\Rightarrow\,(\wedge\,(\texttt{beautiful}\,(x)\,, \subseteq\,(y, x))\,, \texttt{beautiful}\,(y)))$$

Finally, we remark on a few additional features of the MMT language that are important for large-scale case studies but not critical to understand the basic intuitions of results. MMT provides a module system that allows theories to instantiate and import each other. The module system is conservative: every theory can be *elaborated* into one that only declares constants. MMT constants may carry an optional definiens, in which case we write $c : E = e$. Defined constants can be eliminated by definition expansion.

3 Intra-Library View Finding

viewfinding

Let C be a corpus of theories with the same fixed meta-theory M. We call the problem of finding theory views between theories of C the **view finding problem** and an algorithm that solves it a **view finder**. Note that a view finder is sufficient to solve the theory classification use case from the introduction: Jane provides a M-theory Q of beautiful sets, the view finder computes all (total) views from Q into C.

Efficiency Considerations The cost of this problem quickly explodes. First of all, it is advisable to restrict attention to simple views. Eventually we want to search for arbitrary views as well. But that problem is massively harder because

it subsumes theorem proving: a view from Σ to Σ' maps Σ-axioms to Σ'-proofs, i.e., searching for a view requires searching for proofs.

Secondly, if C has n theories, we have n^2 pairs of theories between which to search. (It is exactly n^2 because the direction matters, and even views from a theory to itself are interesting.) Moreover, for two theories with m and n constants, there are n^m possible simple views. (It is exactly n^m because views may map different constants to the same one.) Thus, we can in principle enumerate and check all possible simple views in C. But for large C, it quickly becomes important to do so in an efficient way that eliminates ill-typed or uninteresting views early on.

Thirdly, it is desirable to search for *partial* views as well. In fact, identifying refactoring potential in libraries is only possible if we find partial views: then we can refactor the involved theories in a way that yields a total view. Moreover, many proof assistant libraries do not follow the little theories paradigm or do not employ any theory-like structuring mechanism at all. These can only be represented as a single huge theory, in which case we have to search for partial views from this theory to itself. While partial views can be reduced to and then checked like total ones, searching for partial views makes the number of possible views that must be checked much larger.

Finally, even for a simple view, checking reduces to a set of equality constraints, namely the constraints $\vdash_{\Sigma'} \overline{\sigma}(E) = E'$ for the type-preservation condition. Depending on M, this equality judgment may be undecidable and require theorem proving.

Algorithm Overview A central motivation for our algorithm is that equality in M can be soundly approximated very efficiently by using a normalization function on M-expressions. This has the additional benefit that relatively little meta-theory-specific knowledge is needed, and all such knowledge is encapsulated in a single well-understood function. This way we can implement view–search generically for arbitrary M.

Our algorithm consists of two steps. First, we preprocess all constant declarations in C with the goal of moving as much intelligence as possible into a step whose cost is linear in the size of C. Then, we perform the view search on the optimized data structures produced by the first step.

3.1 Preprocessing

The preprocessing phase computes for every constant declaration $c : E$ a normal form E' and then efficiently stores the abstract syntax tree of E'. Both steps are described below.

Normalization involves two steps: **MMT-level normalization** performs generic transformations that do not depend on the meta-theory M. These include elaboration of structured theories and definition expansion, which we mentioned in Sect. 2. Importantly, we do not fully eliminate defined constant declarations $c : E = e$ from a theory Σ: instead, we replace them with primitive constants

$c : E$ and replace every occurrence of c in other declarations with e. If Σ is the domain theory, we can simply ignore $c : E$ (because views do not have to provide an assignment to defined constants). But if the Σ is the codomain theory, retaining $c : E$ increases the number of views we can find; in particular in situations where E is a type of proofs, and hence c a theorem.

Meta-theory-level normalization applies an M-specific normalization function. In general, we assume this normalization to be given as a black box. However, because many practically important normalization steps are widely reusable, we provide a few building blocks, from which specific normalization functions can be composed. Skipping the details, these include:

1. Top-level universal quantifiers and implications are rewritten into the function space of the logical framework using the Curry-Howard correspondence.
2. The order of curried domains of function types is normalized as follows: first all dependent arguments types ordered by the first occurrence of the bound variables; then all non-dependent argument types A ordered by the abstract syntax tree of A.
3. Implicit arguments, whose value is determined by the values of the others are dropped, e.g. the type argument of an equality. This has the additional benefit or shrinking the abstract syntax trees and speeding up the search.
4. Equalities are normalized such that the left hand side has a smaller abstract syntax tree.

Above multiple normalization steps make use of a total order on abstract syntax trees. We omit the details and only remark that we try to avoid using the names of constants in the definition of the order — otherwise, declarations that could be matched by a view would be normalized differently. Even when breaking ties between requires comparing two constants, we can first try to recursively compare the syntax trees of their types.

abstractsyntaxtree

Abstract Syntax Trees We define **abstract syntax trees** as pairs (t, s) where t is subject to the grammar

$$t ::= C_{Nat} \mid V_{Nat} \mid t\,[t^+]\,(t^+)$$

(where Nat is a non-terminal for natural numbers) and s is a list of constant names.

We obtain an abstract syntax tree from an MMT expression E by (i) switching to de-Bruijn representation of bound variables and (ii) replacing all occurrences of constants with C_i in such a way that every C_i refers to the i-th element of s.

Abstract syntax trees have the nice property that they commute with the application of simple views σ: If (t, s) represents E, then $\sigma(E)$ is represented by (t, s') where s' arises from s by replacing every constant with its σ-assignment.

The above does not completely specify i and s yet, and there are several possible canonical choices among the abstract syntax trees representing the same expression. The trade-off is subtle because we want to make it easy to both identify and check views later on. We call (t, s) the **long** abstract syntax tree for E if C_i replaces the i-th occurrence of a constant in E when E is read in left-

to-right order. In particular, the long tree does not merge duplicate occurrences of the same constant into the same number. The **short** abstract syntax tree for E arises from the long one by removing all duplicates from s and replacing the C_i accordingly.

Example 1. Consider again the axiom $\forall x : \mathtt{set}, y : \mathtt{set}.\, \mathtt{beautiful}(x) \wedge y \subseteq x \Rightarrow \mathtt{beautiful}(y)$ with internal representation

$$\forall\left[x : \mathtt{set}, y : \mathtt{set}\right]\left(\Rightarrow\left(\wedge\left(\mathtt{beautiful}\left(x\right), \subseteq\left(y, x\right)\right), \mathtt{beautiful}\left(y\right)\right)\right).$$

The *short* syntax tree and list of constants associated with this term would be:

$$t = C_1\left[C_2, C_2\right]\left(C_3\left(C_4\left(C_5\left(V_2\right), C_6\left(V_1, V_2\right)\right), C_5\left(V_1\right)\right)\right)$$

$$s = \left(\forall, \mathtt{set}, \Rightarrow, \wedge, \mathtt{beautiful}, \subseteq\right)$$

The corresponding long syntax tree is :

$$t = C_1\left[C_2, C_3\right]\left(C_4\left(C_5\left(C_6\left(V_2\right), C_7\left(V_1, V_2\right)\right), C_8\left(V_1\right)\right)\right)$$

$$s = \left(\forall, \mathtt{set}, \mathtt{set}, \Rightarrow, \wedge, \mathtt{beautiful}, \subseteq, \mathtt{beautiful}\right)$$

For our algorithm, we pick the *long* abstract syntax tree, which may appear surprising. The reason is that shortness is not preserved when applying a simple view: whenever a view maps two different constants to the same constant, the resulting tree is not short anymore. Length, on the other hand, is preserved. The disadvantage that long trees take more time to traverse is outweighed by the advantage that we never have to renormalize the trees.

3.2 Search

Consider two constants $c : E$ and $c' : E'$, where E and E' are preprocessed into long abstract syntax trees (t, s) and (t', s'). It is now straightforward to show the following Lemma:

Lemma 1. *The assignment $c \mapsto c'$ is well-typed in a view σ if $t = t'$ (in which case s and s' must have the same length l) and σ also contains $s_i \mapsto s'_i$ for $i = 1, \ldots, l$.*

Of course, the condition about $s_i \mapsto s'_i$ may be redundant if s contain duplicates; but because s has to be traversed anyway, it is cheap to skip all duplicates. We call the set of assignments $s_i \mapsto s'_i$ the **prerequisites** of $c \mapsto c'$.

This lemma is the center of our search algorithm explained in

Lemma 2 (Core Algorithm). *Consider two constant declarations c and c' in theories Σ and Σ'. We define a view by starting with $\sigma = c \mapsto c'$ and recursively adding all prerequisites to σ until*

- *either the recursion terminates*
- *or σ contains two different assignments for the same constant, in which case we fail.*

If the above algorithm succeeds, then σ is a well-typed partial simple view from Σ to Σ'.

Example 2. Consider two constants c and c' with types $\forall x : \texttt{set}, y : \texttt{set}.\ \texttt{beautiful}(x) \wedge y \subseteq x \Rightarrow \texttt{beautiful}(y)$ and $\forall x : \texttt{powerset}, y : \texttt{powerset}.\ \texttt{finite}(x) \wedge y \subseteq x \Rightarrow \texttt{finite}(y)$. Their syntax trees are

$$t = t' = C_1\left[C_2, C_3\right]\left(C_4\left(C_5\left(C_6\left(V_2\right), C_7\left(V_1, V_2\right)\right), C_8\left(V_1\right)\right)\right)$$

$$s = (\forall, \texttt{set}, \texttt{set}, \Rightarrow, \wedge, \texttt{beautiful}, \subseteq, \texttt{beautiful})$$

$$s' = (\forall, \texttt{powerset}, \texttt{powerset}, \Rightarrow, \wedge, \texttt{finite}, \subseteq, \texttt{finite})$$

Since $t = t'$, we set $c \mapsto c'$ and compare s with s', meaning we check (ignoring duplicates) that $\forall \mapsto \forall$, $\texttt{set} \mapsto \texttt{powerset}$, $\Rightarrow\mapsto\Rightarrow$, $\wedge \mapsto \wedge$, $\texttt{beautiful} \mapsto \texttt{finite}$ and $\subseteq\mapsto\subseteq$ are all valid.

To find all views from Σ to Σ', we first run the core algorithm on every pair of Σ-constants and Σ'-constants. This usually does not yield big views yet. For example, consider the typical case where theories contain some symbol declarations and some axioms, in which the symbols occur. Then the core algorithm will only find views that map at most one axiom.

Depending on what we intend to do with the results, we might prefer to consider them individually (e.g. to yield *alignments* in the sense of [Kal+16]). But we can also use these small views as building blocks to construct larger, possibly total ones:

amalgamatingviews

Lemma 3 (Amalgamating Views). *We call two partial views **compatible** if they agree on all constants for which both provide an assignment.*

The union of compatible well-typed views is again well-typed.

Example 3. Consider the partial view from Example 2 and imagine a second partial view for the axioms $\texttt{beautiful}(\varnothing)$ and $\texttt{finite}(\varnothing)$. The former has the requirements

$$\forall \mapsto \forall, \quad \texttt{set} \mapsto \texttt{powerset} \quad \Rightarrow\mapsto\Rightarrow \quad \wedge \mapsto \wedge \quad \texttt{beautiful} \mapsto \texttt{finite} \quad \subseteq\mapsto\subseteq$$

The latter requires only $\texttt{set} \mapsto \texttt{powerset}$ and $\varnothing \mapsto \varnothing$. Since both views agree on all assignments, we can merge all oo them them into a single view, mapping both axioms and all requirements of both.

3.3 Optimizations

The above presentation is intentionally simple to convey the general idea. We now describe a few advanced features of our implementation to enhance scalability.

Caching Preprocessing Results Because the preprocessing performs normalization, it can be time-consuming. Therefore, we allow for storing the preprocessing results to disk and reloading them in a later run.

Fixing the Meta-Theory We improve the preprocessing in a way that exploits the common meta-theory, which is meant to be fixed by every view. All we have to do is, when building the abstract syntax trees (t, s), to retain all references to constants of the meta-theory in t instead of replacing them with numbers. With this change, s will never contain meta-theory constants, and the core algorithm will only find views that fix all meta-theory constants. Because s is much shorter now, the view search is much faster.

It is worth pointing out that the meta-theory is not always as fixed as one might think. Often we want to consider to be part of the meta-theory certain constants that are defined early on in the library and then used widely. In PVS, this makes sense, e.g., for all operations define in the Prelude library (the small library shipped with PVS). Note that we still only have to cache one set of preprocessing results for each library: changes to the meta-theory only require minor adjustments to the abstract syntax trees without redoing the entire normalization.

Biasing the Core Algorithm The core algorithm starts with an assignment $c \mapsto c'$ and then recurses into constant that occur in the declarations of c and c'. This occurs-in relation typically splits the constants into layers. A typical theory declares types, which then occur in the declarations of function symbols, which then occur in axioms. Because views that only map type and function symbols are rarely interesting (because they do not allow transporting non-trivial theorems), we always start with assignments where c is an axiom.

Exploiting Theory Structure Libraries are usually highly structured using imports between theories. If Σ is imported into Σ', then the set of partial views out of Σ' is a superset of the set of partial views out of Σ. If implemented naively, that would yield a quadratic blow-up in the number of views to consider.

Instead, when running our algorithm on an entire library, we only consider views between theories that are not imported into other theories. In an additional postprocessing phase, the domain and codomain of each found partial view σ are adjusted to the minimal theories that make σ well-typed.

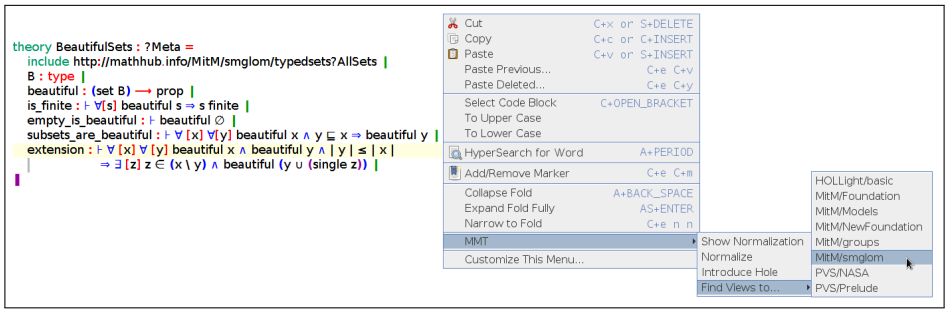

Fig. 2. "Beautiful Sets" in MMT Surface Syntax

3.4 Implementation

We have implemented our view finder algorithm in the MMT system. A screenshot of Jane's theory of beautiful sets is given in Figure 2. Right-clicking anywhere within the theory allows Jane to select `MMT` → `Find Views to...` → `MitM/smglom`. The latter menu offers a choice of known libraries in which the view finder should look for codomain theories; `MitM/smglom` is the Math-in-the-Middle library based that we have developed [Deh+16] to describe the common knowledge used in various CICM systems.

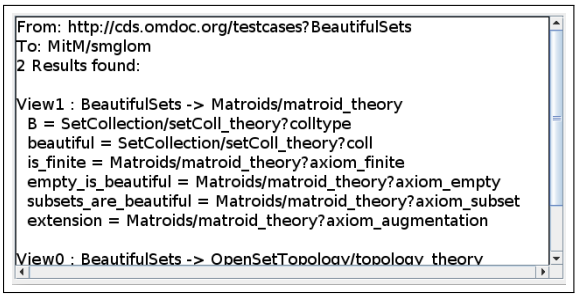

Fig. 3. Views found for "Beautiful Sets"

```
theory Matroids : base:?Logic =
  include ?Properties |
  include typedsets?FiniteCardinality |
  include arithmetics?NaturalArithmetics |

  theory matroid_theory : base:?Logic > X : type | =
    include typedsets?SetCollection/setColl_theory (X) |

    axiom_finite : {s : set X} ⊢ s ∈ coll ⟶ ⊢ s finite |
    axiom_empty : ⊢ prop_emptyInColl coll |
    axiom_subset : ⊢ prop_subsetClosed coll |
    axiom_augmentation : ⊢ ∀ [A] ∀ [B] A ∈ coll ∧ B ∈ coll ∧ | B | ≤ | A |
        ⇒ ∃ [x] x ∈ (A \ B) ∧ (B ∪ (single x)) ∈ coll |
  |
|
```

Fig. 4. The Theory of Matroids in the MitM Library

After choosing `MitM/smglom`, the view finder finds two views (within less than one second) and shows them (Figure 3). The first of these (`View1`) has a theory for matroids as its codomain, which is given in Figure 4. Inspecting that theory and the assignments in the view, we see that it indeed represents the well-known correspondence between beautiful sets and matroids.

4 Inter-Library View Finding

We now generalize to view finding to different libraries written in different logics. Intuitively, the key idea is that we now have two fixed meta-theories M and M' and a fixed meta-view $m : M \to M'$. However, due to the various idiosyncrasies of logics, tools' library structuring features, individual library conventions, this problem is significantly more difficult than *intra*-library view finding. For example, unless the logics are closely related, meta-views usually do not even exist and must be approximated. Therefore, a lot of tweaking is typically necessary, and it is possible that multiple runs with different trade-offs give different interesting results.

As an example, we present a large case study where we find views from the MitM library used in the running example so far into the PVS/NASA library.

4.1 The PVS/NASA Library

PVS [ORS92] is a proof assistant under active development based on a higher-order logic with a number of advanced features. In addition to the *Prelude* library, which contains the most common domains of mathematical discourse and is shipped with PVS itself, there is a large library of formal mathematics developed and maintained by NASA [PVS]. In [Koh+17], we represent PVS as a meta-theory in MMT and implemented a translator that transforms both libraries into MMT format. We use a meta-view that embeds MitM's higher-order logic into PVS's higher-order logic and make sure that we normalize PVS-formulas in the same way as MitM-formulas.

Theory Structure Normalization PVS's complex and prevalently used parametric theories critically affect view finding because they affect the structure of theories. For example, the theory of groups `group_def` in the NASA library has three theory parameters $(\texttt{T}, *, \texttt{one})$ for the signature of groups, and includes the theory `monoid_def` with the same parameters, and then declares the axioms for a group in terms of these parameters. Without special treatment, we could only find views from/into libraries that use the same theory structure.

We have investigated three approaches of handling parametric theories:

1. *Simple treatment:* We drop theory parameters and interpret references to them as free variables that match anything. This is of course not sound so that all found views must be double-checked. However, because practical search problems often do not require exact results, even returning all potential views can be useful.
2. *Covariant elimination:* We treat theory parameters as if they were constants declared in the body. In the above mentioned theory `group_def`, we would hence add three new constants `T`, $*$ and `one` with their corresponding types. This works well in the common case where a parametric theory is not used with two different instantiations in the same context.
3. *Contravariant elimination:* The theory parameters are treated as if they were bound separately for every constant in the body of the theory. In the above mentioned theory `group_def`, we would change e.g. the unary predicate `inverse_exists?` with type $T \to \texttt{bool}$ to a function with type $(T : \texttt{pvstype}) \to (* : T \to T \to T) \to (\texttt{one} : T) \to (T \to \texttt{bool})$. This is closest to the actual semantics of the PVS module system. But it makes finding interesting views the least likely because it is the most sensitive to the modular structure of individual theories.

We have implemented the first two approaches. The first is the most straightforward but it leads to many false positives and false negatives. We have found the second approach to be the most useful for inter-library search since it most closely corresponds to simple formalizations of abstract theories in other libraries. The third approach will be our method of choice when investigating *intra*-library views of PVS/NASA in future work.

4.2 Implementation

As a first use case, we can write down a theory for a commutative binary operator using the MitM foundation, while targeting the PVS Prelude library – allowing us to find all commutative operators, as in Figure 5 (using the simple approach to theory parameters).

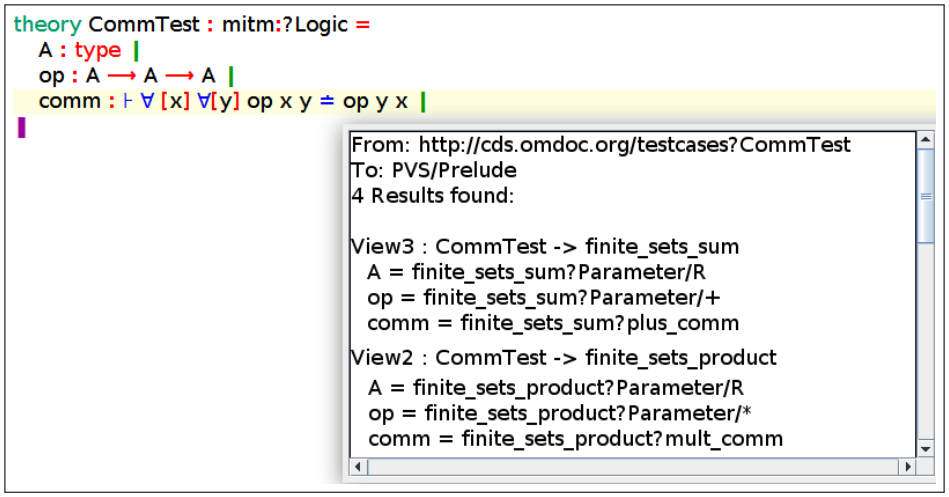

Fig. 5. Searching for Commutative Operators in PVS

This example also hints at a way to iteratively improve the results of the view finder: since we can find properties like commutativity and associativity, we can use the results to in turn inform a better normalization of the theory by exploiting these properties. This in turn would potentially allow for finding more views.

To evaluate the approaches to theory parameters we used a simple theory of monoids in the MitM foundation and the theory of monoids in the NASA library as domains for viewfinding with the whole NASA library as target using simple and covariant approaches. The results are summarized in Figure 6.

Domain	Normalization	Simple Views	Aggregated
NASA/monoid	simple	388	154
MitM/monoid	simple	32	17
NASA/monoid	covariant	1026	566
MitM/monoid	covariant	22	6

Fig. 6. Results of Inter- and Intra-Library View Finding in the PVS NASA Library

Most of the results in the simple MitM→NASA case are artifacts of the theory parameter treatment and view amalgamation – in fact only two of the 17 results are meaningful (to operations on sets and the theory of number fields). In the covariant case, the additional requirements lead to fuller (one total) and less spurious views. With a theory from the NASA library as domain, the results are already too many to be properly evaluated by hand. With the simple approach to theory parameters, most results can be considered artifacts; in the covariant case, the most promising results yield (partial) views into the theories of semigroups, rings (both the multiplicative and additive parts) and most extensions thereof (due to the duplication of theory parameters as constants).

5 Conclusion

We present a general MKM utility that given a MMT theory and an MMT library $\mathcal{L}$ finds partial and total views into $\mathcal{L}$. Such a view finder can be used to drive various MKM applications ranging from theory classification to library merging and refactoring. The theory discovery use case described in Sect. 3.4 is mostly desirable in a setting where a user is actively writing or editing a theory, so the integration in jEdit is sensible. However, the inter-library view finding would be a lot more useful in a theory exploration setting, such as when browsing available archives on MathHub [Ian+14] or in the graph viewer integrated in Mmt [RKM17].

Future Work The current view finder is already efficient enough for the limited libraries we used for testing. To increase efficiency, we plan to explore term indexing techniques [Gra96] that support $1 : n$ and even $n : m$ matching and unification queries. The latter will be important for the library refactoring and merging applications which look for all possible (partial and total) views in one or between two libraries. As such library-scale operations will have to be run together with theory flattening to a fixed point and re-run upon every addition to the library, it will be important to integrate them with the MMT build system and change management processes [AM10; Ian12].

Enabled Applications Our work enables a number of advanced applications. Maybe surprisingly, a major bottleneck here concerns less the algorithm or software design challenges but user interfaces and determining the right application context.

- **Model-/Countermodel Finding:** If the codomain of a view is a theory representing a specific model, it would tell Jane that those are *examples* of her abstract theory. Furthermore, partial views – especially those that are total on some included theory – could lead to insightful *counterexamples*.
- **Library Refactoring:** Given that the view finder looks for *partial* views, we can use it to find natural extensions of a starting theory. Imagine Jane removing the last of her axioms for "beautiful sets" – the other axioms (disregarding finiteness of her sets) would allow her to find e.g. both Matroids

and *Ideals*, which would suggest to her to possibly refactor her library such that both extend her new theory. Additionally, *surjective* partial views would inform her, that her theory would probably better be refactored as an extension of the codomain, which would allow her to use all theorems and definitions therein.
- **Theory Generalization:** If we additionally consider views into and out of the theories found, this can make theory discovery even more attractive. For example, a view from a theory of vector spaces intro matroids could inform Jane additionally, that her beautiful sets, being matroids, form a generalization of the notion of linear independence in linear algebra.
- **Folklore-based Conjecturing:** If we have theory T describing (the properties of) a class O of objects under consideration and a view $v : S \nrightarrow T$, then we can use extensions of S' in $\mathcal{L}$ with $\iota : S \hookrightarrow S'$ for making conjectures about O: The v-images of the local axioms of S' would make useful properties to establish about O, since they allow pushing out v over ι to a view $v' : S' \nrightarrow T'$ (where T' extends T by the newly imported properties) and gain $v'(S')$ as properties of O. Note that we would need to keep book on our transformations during preprocessing and normalization, so that we could use the found views for translating both into the codomain as well as back from there into our starting theory. A useful interface might specifically prioritize views into theories on top of which there are many theorems and definitions that have been discovered.

Note that even though the algorithm is in principle symmetric, some aspects often depend on the direction — e.g. how we preprocess the theories, which constants we use as starting points or how we aggregate and evaluate the resulting (partial) views (see Sections 3.3, 3.1 and 4.1).

Acknowledgments The authors gratefully acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541) and the DFG-funded project OAF: An Open Archive for Formalizations (KO 2428/13-1).

References

[AM10] S. Autexier and N. Müller. "Semantics-based Change Impact Analysis for Heterogeneous Collections of Documents". In: *Proceedings of the 10th ACM symposium on Document engineering*. Ed. by M. Gormish and R. Ingold. DocEng '10. Manchester, United Kingdom: ACM, 2010, pp. 97–106. DOI: http://doi.acm.org/10.1145/1860559.1860580.

[Deh+16] P. Dehaye et al. "Interoperability in the ODK Project: The Math-in-the-Middle Approach". In: *Intelligent Computer Mathematics*. Ed. by M. Kohlhase, L. de Moura, M. Johansson, B. Miller, and F. Tompa. Springer, 2016, pp. 117–131.

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

[GK14] T. Gauthier and C. Kaliszyk. "Matching concepts across HOL libraries". In: *Intelligent Computer Mathematics*. Ed. by S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban. Springer, 2014, pp. 267–281.

[Gra96] P. Graf. *Term Indexing*. LNCS 1053. Springer Verlag, 1996.

[Ian+14] M. Iancu, C. Jucovschi, M. Kohlhase, and T. Wiesing. "System Description: MathHub.info". In: *Intelligent Computer Mathematics*. Ed. by S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban. Springer, 2014, pp. 431–434.

[Ian12] M. Iancu. "Management of Change in Declarative Languages". MA thesis. Jacobs University Bremen, 2012.

[Kal+16] C. Kaliszyk, M. Kohlhase, D. Müller, and F. Rabe. "A Standard for Aligning Mathematical Concepts". In: *Work in Progress at CICM 2016*. Ed. by A. Kohlhase, M. Kohlhase, P. Libbrecht, B. Miller, F. Tompa, A. Naummowicz, W. Neuper, P. Quaresma, and M. Suda. CEUR-WS.org, 2016, pp. 229–244.

[Koh+17] M. Kohlhase, D. Müller, S. Owre, and F. Rabe. "Making PVS Accessible to Generic Services by Interpretation in a Universal Format". In: *Interactive Theorem Proving*. Ed. by M. Ayala-Rincon and C. Munoz. Springer, 2017, pp. 319–335.

[Koh06] M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Lecture Notes in Artificial Intelligence 4180. Springer, 2006.

[MRK] D. Müller, F. Rabe, and M. Kohlhase. *Automatically Finding Theory Morphisms for Knowledge Management*. URL: http://kwarc.info/kohlhase/submit/viewfinder-report.pdf.

[MWP] *Matroid — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Matroid (visited on 04/04/2018).

[NK07] I. Normann and M. Kohlhase. "Extended Formula Normalization for ε-Retrieval and Sharing of Mathematical Knowledge". In: *Towards Mechanized Mathematical Assistants. MKM/Calculemus*. Ed. by M. Kauers, M. Kerber, R. Miner, and W. Windsteiger. LNAI 4573. Springer Verlag, 2007, pp. 266–279.

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

[Pau94] L. Paulson. *Isabelle: A Generic Theorem Prover*. Vol. 828. Lecture Notes in Computer Science. Springer, 1994.

[PVS] *NASA PVS Library*. URL: http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/ (visited on 12/17/2014).

[RK13] F. Rabe and M. Kohlhase. "A Scalable Module System". In: *Information and Computation* 230.1 (2013), pp. 1–54.

[RKM17] M. Rupprecht, M. Kohlhase, and D. Müller. "A Flexible, Interactive Theory-Graph Viewer". In: *MathUI 2017: The 12th Workshop on Mathematical User Interfaces*. Ed. by A. Kohlhase and M. Pollanen. 2017. URL: http://kwarc.info/kohlhase/papers/mathui17-tgview.pdf.