Extended Formula Normalization for ϵ-Retrieval and Sharing of Mathematical Knowledge

Immanuel Normann and Michael Kohlhase

Computer Science, Jacobs University Bremen
{i.normann,m.kohlhase}@jacobs-university.de

Abstract. Even though only a tiny fraction of mathematical knowledge is available digitally (e.g in theorem prover or computer algebra libraries, in documents with content-markup), our current retrieval methods are already inadequate. With further increase in digitalization of mathematics the situation will get worse without significant advances.

When searching a formula, we often want to find not only structurally identical occurrences, but also all (logically) equivalent ones. Furthermore, we want to retrieve whole mathematical theories (i.e. objects with prescribed properties), and we want to find them irrespective of the nomenclature chosen in the respective formalization.

In this paper, we propose a normalization-based approach to mathematical formula and theory retrieval modulo an equivalence theory and concept renaming, and apply the proposed algorithm to end-user querying and knowledge sharing. We test the implementation by applying it to a first-order translation of the Mizar library.

1 Introduction

The last two decades have seen a slow but steady accumulation of formalizations (or at least content-representation) of mathematical knowledge. We have the Mizar Mathematical library [Rud92, Miz] with over 40 000 theorems, definitions, and proofs, as well as the the PVS [ORS92, PVS], NuPRL [CAB+86, Nup], and Coq [Tea] libraries of comparable size. But the developments tend to be system-specific, non-interoperable, and redundant. Even inside a single library, it is often simpler to reprove a theorem than finding an equivalent or stronger one to reference. At the level of mathematical theories, the problem is aggravated, since making theories applicable usually involves renaming (or reinterpreting) vocabularies. In this area the field of mathematical knowledge management (MKM) has not yet delivered its initial promise, i.e. that an investment into formalization (or content markup) would yield improvements in automated management. The process of the (human) mathematical community, which is based on peer-review, communication, understanding and reformulating mathematical theories seems to deliver more theory-reuse than the MKM-based counterpart. We believe that to change this, we must solve the *knowledge retrieval* problem at the heart of finding applicable theorems and theory reuse. It is here that our current technology is inadequate. To find a theorem to refer to in a library we should not

M. Kauers et al. (Eds.): MKM/Calculemus 2007, LNAI 4573, pp. 356–370, 2007.
© Springer-Verlag Berlin Heidelberg 2007

have to know its exact mathematical structure, or even its visual appearance[1]. Consider for instance three of many more possible variants to formalize "f is a continuous function":

$$\begin{array}{l}\forall\varepsilon.\varepsilon > 0 \Rightarrow \exists\delta.\forall x.\forall y.0 < |x-y| \wedge |x-y| < \delta \Rightarrow |f(x)-f(y)| < \varepsilon \\ \forall\varepsilon.\exists\delta.\forall x,y.\varepsilon > 0 \Rightarrow (0 < |x-y| \wedge |x-y| < \delta \Rightarrow |f(x)-f(y)| < \varepsilon) \\ \forall\varepsilon.\exists\delta.\forall x,y.\varepsilon > 0 \wedge |x-y| < \delta \wedge 0 < |x-y| \Rightarrow |f(x)-f(y)| < \varepsilon\end{array} \quad (1)$$

A query for one should also find the others, since they are all equivalent. Given an equality theory E, we speak of an **E-retrieval** task, if we want to find all occurrences of formulae t in a collection that are E-equal to a query q. If the theory E includes logical equivalence, we speak of an ϵ-retrieval task. Unfortunately semantical equivalence is undecidable in general, so we will have to make due with search engines that satisfies the query intention approximatively. The best approximation could be achieved by enlisting automated theorem provers during search, but performance and termination problems make this approach intractable, so we have to restrict our notion of equivalence. There is also another reason to restrict equivalence during retrieval: under full logical equivalence, all tautologies become equivalent, since they are all valid.

Rather than coming up with a theoretical solution of relaxed equivalence, we will integrate approximation into the retrieval process itself to ensure efficiency. We view logical equivalence as an equality theory and relax it by viewing two formulae as equivalent, iff they can be co-normalized. Given a normalization function ν, we say that the formulae are **ν-equivalent**, iff their ν normal forms are syntactically identical. ν-equivalence provides an efficient and useful notion of equivalence[2], efficiency in **$\nu\epsilon$-retrieval** (i.e. ϵ-retrieval modulo ν-equivalence) is achieved by normalizing the search corpus of the digital library at indexing time. This moves the cost of ν-retrieval almost completely to the preprocessing/indexing phase: normalizing the query formulae is cheap, since they are usually rather small.

We will consider two applications of $\nu\epsilon$-retrieval that need two slightly different forms of normalization in this paper. In $\nu\epsilon$-retrieval for formula search — e.g. to extend the MathWebSearch formula search engine [Kc06, Kc07] — we are interested in a ν-retrieval of formulae with given constants (e.g. the absolute value function in the formulae in (1)). If we want to $\nu\epsilon$-retrieve whole mathematical theories (i.e. objects with prescribed properties), we want to find them irrespective of the nomenclature chosen in the respective formalization. We will concentrate on the latter application in this paper, since it is the more radical application and indicate where we need to make modifications to the normalization process for the former.

An application of $\nu\epsilon$-retrieval, which we will not cover in this paper is library merging. Currently, formalized mathematical content is scattered across many

[1] Which can differ considerably, even if the structure is known. Consider for instance the presentational variants $\forall x.e^x > 0$ as equivalent to $\forall t.\exp(t) > 0$.

[2] Semantical formula search machines as MBase and MathWebSearch are optimized on speed performance by supporting only alphabetical renaming of bound variables efficiently.

digital libraries which overlap considerably; e.g. one can find a basic Algebra concept like "group" in almost every digital math library. Merging these libraries would obviously improve the knowledge accessibility since redundant searches could be avoided. More importantly, however, knowledge could be enlarged if theory inclusions can be detected.

2 Formula Normalization for Equivalence

This section introduces all normalizers used in this paper. Recall that in term rewriting we say that a formula (or term) is in $\mathcal{R}$-*normal form* iff it is not reducible w.r.t. a given set $\mathcal{R}$ of rewrite rules. $\mathcal{R}$ is called *normalizing* iff every input formula has a $\mathcal{R}$-normal form. Note that normalizing rewrite rules do not necessarily produce unique normal forms as they depend on the application order. Any terminating choice of application order yields unique normal forms and thus a formula transformation mapping. Note furthermore, that any normalizing algorithm f is idempotent, i.e. $f(f(t)) = f(t)$ for any formula t. For the purposes of this paper, we need to generalize: We will call a formula transformation mapping a **normalizer** iff it is idempotent and preserves the semantics of formulae. Normalizer given by a set $\mathcal{R}$ of rewrite rules, are prime examples, but we will also encounter others below.

The methods described in this paper are largely independent of the choice of particular logic, we will presuppose a standard formulation of first-order logic with the quantifiers $\forall$ and $\exists$. We will use a kind of vector notation on quantification binders, where $\overline{x}$ stands for a sequence $x_1, \ldots, x_n$ of variables. Contrary to standard expositions of first-order logic, we will ruthlessly use numbers as variable-, constant-, function- and predicate symbols in the normalization process. These should be considered as purely presentational devices for an infinite, ordered supply of (traditional) symbols of the respective arities. Let us explicitly note that all methods presented in this paper work with typed and higher-order logics with little change. In extensional higher-order logics, where propositions can be embedded into terms more care must be taken to ensure that the embedded propositions are also normalized. Moreover, logics with sequential logical operators as used e.g. in PVS [ORS92, PVS] (where in a formula $A \wedge B$, the subformula B will never be evaluated and might even be ill-typed if A can be determined to be false) will need some adaption.

The most prominent normalizers involved here are the various transformations to negation-, prenex-, and conjunctive/disjunctive-normal form. Their rewrite rules can be found in introductory text books on logic. Next we introduce two straightforward but useful normalizers:

Merge Bindings. The *merge bindings normalizer* ∇_{mb} is defined by the rewrite rule $Q\overline{x}.Q\overline{y}.\varphi \to Q\overline{xy}.\varphi$ for both quantifiers $Q \in \{\forall, \exists\}$.

Sort Binders. The normalizer *sort binders* (∇_{sb}) is defined as procedure lexicographically sorting all the binders of a formula, e.g. $\nabla_{\mathrm{sb}}(\forall z, w.\exists y, x, z.\phi) = \forall w, z.\exists x, y, z.\phi$.

The following sections contain some normalizers that — even though they seem simple — seem not to have been discussed in the literature yet even though AC-normalization must have been considered (and rejected) for AC-unification. Incidentally, neither of the solutions presented here were found in a recent related article about "Semantic Matching for Mathematical Services" [NP05]. At least the *binding-last* strategy is an improvement over the folklore solution.

2.1 α-Normalization

An **α-normalizer** is a formula transformation mapping that replaces each non-logical symbol occurrence in a formula φ by a number where different occurrences of equal variables are substituted by the same number, but different variables by different numbers. Obviously this procedure is idempotent and always terminating. Moreover renaming bound variables does not change the semantics of a formula: input and output formulae are said to be **α-equivalent**. Renaming free variables or constants does not change the semantics of a formula provided that we rename consistently all involved free variables and constants of the context where the formula lives (cf. Section 3). Depending on the application, the notion of what constitutes a context may change, and we assume a representation that makes contexts explicit, e.g. as nested contexts in the Coq libraries [Tea] or in OMDoc documents [Koh06]. For our theory-retrieval application, the context will be a theory.

Let ∇_{as} be the α-normalizer induced by a depth-first but binding-last order. The latter means that the numbering of binding occurrences is executed after the numbering of the formula body is finished. For instance take the formula $\forall x, y.R(f(x, y), x)$ then the procedure is as follows:

1. The depth-first walk through the body of the formula builds the renaming $\sigma = [R \to 1, f \to 2, x \to 3, y \to 4]$ and simultaneously applies it to the body which yields $\forall x, y.\underline{1(2(3, 4), 3)}$.
2. The renaming σ is applied to the binding occurrences: $\forall \underline{3, 4}.1(2(3, 4), 3)$

As a renaming of α-normalization is bijective we can always easily reconstruct the original formula. To make the explicit notation of renaming more convenient we make use of this bijectivity: Instead of $\sigma = [v_1 \to 1, \ldots, v_n \to n]$ we will just write $\sigma = [v_1, \ldots, v_n]$.

Binding-last is Superior to Binding-first Numbering. Naive α-normalization does binding-first numbering; i.e. the numbering of the variables is determined by the order in binding-occurrence. To see the advantage of the binding-last strategy, consider the following equivalent formulae $t_1, \ldots, t_4$:

$$\forall x, y.R(x, y) \approx \forall x, y.R(y, x) \approx \forall y, x.R(x, y) \approx \forall y, x.R(y, x)$$

If we apply binding-first numbering and ∇_{sb} to them, we obtain two normal forms irrespective of the order — e.g.

$$\forall 1, 2.3(1, 2) \text{ for } t_1 \text{ and } t_3 \quad \text{and} \quad \forall 1, 2.3(2, 1) \text{ for } t_2 \text{ and } t_4$$

for ∇_{sb} before numbering. But $\nabla_{\text{sb}}(\nabla_{\text{as}}(t)) = \forall 2, 3.1(2, 3)$ for all four formulae!

2.2 Normalization for ACI Operators

Consider two applications $f(x_1, \ldots, x_n)$ and $f(y_1, \ldots, y_n)$ where f is an associative, commutative and idempotent (ACI) operator, e.g. the binary logical operators, set union, or set intersection. For ACI-retrieval the task is to find, a permutation π for the arguments $y_1, \ldots, y_n$ such that $x_i = y_{\pi(i)}$ for $i = 1, \ldots, n$. Matching such expressions cause combinatorial explosion if it is done naively. In fact this is one of the weak point of almost all math search engines known to the authors ([DGH96] being a notable exception).

For ACI normalization the respective expression is translated into an object of nested symbol tuples/sets: Let A be an ACI and N be an non-ACI operator; c an atomic and φ_i any kind of expression. Then we translate inductively:

$$\begin{array}{rcl} c & \longmapsto & c \\ A(\varphi_1, \ldots, \varphi_n) & \longmapsto & \langle A, \{\varphi_1, \ldots, \varphi_n\}\rangle \\ N(\varphi_1, \ldots, \varphi_n) & \longmapsto & \langle N, \langle \varphi_1, \ldots, \varphi_n\rangle\rangle \end{array}$$

Obviously every result expression can be uniquely translated back; i.e. the translation is an isomorphic representation of the original formulae. For the following let us call them *ACI representations*. For instance $(a \wedge b) \vee (b \wedge c) \Rightarrow (a \vee c)$ (in prefix notation $\Rightarrow (\vee(\wedge(a, b), \wedge(b, c)), \vee(a, c))$) translates to

$$\langle \Rightarrow, \langle \vee, \{\langle \wedge, \{a, b\}\rangle, \langle \wedge, b, c\rangle\}, \langle \vee, \{a, c\}\rangle\rangle\rangle.$$

We will now define an **ACI normalizer** ∇_{aci} that normalizes the ACI representation of a given expression φ. During the ACI normalization an intermediate ACI expression φ_N and a set of renamings Σ is computed with $\sigma(\pi\varphi) = \varphi_N$ for all $\sigma \in \Sigma$ and an appropriate and semantics preserving permutation π of symbols in φ. Our ACI normal forms after all will be each of $\sigma\varphi$. Most importantly this φ_N will be the minimal expression with respect to a given term ordering. In fact the choice of the term ordering is irrelevant. It just needs to be fixed once, so without loss of generality we use to the following, inductively defined term ordering:

- numbers are ordered as usual, but symbols are considered to be equal.
- A number is smaller than a symbol,
- a symbol is smaller than a tuple,
- for two tuples that with fewer arguments is the smaller
- for two tuples of same arity that tuple is smaller whose first component (going from left to right) is smaller than that of the other tuple, e.g. $\langle 1, 2, \{a, b\}\rangle < \langle 1, 3, 4\rangle$ since $2 < 3$ in the second component (the tail of the tuple doesn't matter).
- a tuple is smaller than a set.
- the comparison between set works similar to that of tuples when sets are represented as ordered tuples.

Note, since symbols are considered to be equal the term ordering of ACI expression is not total in general. However, the set of all ACI expressions where

symbols are replaced by numbers has a total term ordering (since numbers are totally ordered whereas symbols aren not). This is important, because we are looking for the unique representative ACI expression.

The ACI normalization of $\langle\Rightarrow, \langle\vee, \{\langle\wedge, \{a, b\}\rangle, \langle\wedge, b, c\rangle\}, \langle\vee, \{a, c\}\rangle\rangle\rangle$ from our above example returns the normalized ACI expression

$$\langle 0, \langle 1, \{\langle 2, \{3, 4\}\rangle, \langle 2, \{3, 5\}\rangle\}, \langle 1, \{4, 5\}\rangle\rangle\rangle$$

together with these two renamings

$$[imp, \vee, \wedge, b, a, c], \qquad [imp, \vee, \wedge, b, c, a]$$

We now want to sketch our algorithm of ACI normalization: The basic idea is to replace all symbols of the input expression stepwise by numbers augmenting the renamings in parallel with those symbols just removed from the expression. The key idea for that is to remove the next "minimal" symbol from the expression and replace it by a number determined by the current renaming: If this symbol is a member of a renaming then return the corresponding number (see the convention introduced in Section 2.1) and otherwise append the symbol to the renaming and return the corresponding new number.

Let us fortify our intuition with an example and use *prox* for the procedure which fetches all "minimal" symbols from a given expression. Let s be a symbol, n be a number, φ_i be an arbitrary ACI expression then we define inductively:

$$\mathrm{prox}(s) = \{s\} \quad \mathrm{prox}(n) = \emptyset$$

$$\mathrm{prox}(\langle\varphi_1, \varphi_2 \ldots, \varphi_n\rangle) = \begin{cases} \mathrm{prox}(\varphi_1) & \text{if } \varphi_1 \text{ contains symbols} \\ \mathrm{prox}(\langle\varphi_2 \ldots, \varphi_n\rangle) & \text{otherwise} \end{cases}$$

$$\mathrm{prox}(\{\varphi_1 \ldots, \varphi_n\}) = \mathrm{prox}(\varphi_1) \cup \ldots \cup \mathrm{prox}(\varphi_n)$$

For instance *prox* applied on $\{\langle 1, a\rangle, \langle 1, b\rangle\}$ evaluates to:

$$\mathrm{prox}(\{\langle 1, a\rangle, \langle 1, b\rangle\}) \mapsto \mathrm{prox}(\langle 1, a\rangle) \cup \mathrm{prox}(\langle 1, b\rangle) \mapsto \mathrm{prox}(a) \cup \mathrm{prox}(b) \mapsto \{a, b\}$$

The procedure $\min_\sigma$ takes an expression and a set of renamings and filters all renamings that make this expression minimal if applied. For instance $\min_\sigma$ applied on the expression $\varphi := \langle\{a, b\}, \{b, c\}\rangle$ and the renamings $[a, b]$ and $[a, c]$ would return only $[a, b]$. To understand this we have to apply them on φ: $[a, b]\varphi = \langle\{1, 2\}, \{2, c\}\rangle$ and $[a, c]\varphi = \langle\{1, b\}, \{2, c\}\rangle$. The second result expression is greater than the first one. Hence only $[a, b]$ makes φ minimal.

At last we need the $\oplus$ operator that appends a symbol to a renaming; i.e. $[s_1, \ldots, s_n] \oplus s_{n+1} = [s_1, \ldots, s_n, s_{n+1}]$. We extend its definition to arrays of renamings and arrays of symbols by applying $\oplus$ on those components.

With these procedures we write an ACI algorithm as recursive function ∇_{aci} terminating on a fixpoint:

$$\nabla_{\mathrm{aci}}(\Sigma, \varphi) := \begin{cases} \Sigma\varphi & \text{if} \quad \varphi = \nabla_{\mathrm{aci}}(\Sigma, \varphi) \\ \min_\sigma\left(\Sigma \oplus \mathrm{prox}(\Sigma\varphi)\right)(\varphi) & \text{otherwise} \end{cases}$$

Hereby φ is the input ACI expression and Σ a set of renamings evolving along the recursion. Initially Σ is the empty set. The algorithm always terminates since the fixpoint is reached as soon as all symbols of the initial expression φ are replaced by numbers and each recursion step replaces at least one symbol by a number.

For a better understanding we demonstrate the algorithm on the example input $\varphi := \langle\{a,b,c\},\{a,c,d\}\rangle$:

$$\begin{aligned}
\mathrm{prox}(\Sigma\varphi) &= \{a,b,c\} \\
\Sigma \oplus \mathrm{prox}(\Sigma\varphi) &= ([a],[b],[c]) \\
(\Sigma \oplus \mathrm{prox}(\Sigma\varphi))(\varphi) &= \begin{pmatrix} \langle\{1,b,c\},\{1,c,d\}\rangle \\ \langle\{a,1,c\},\{a,c,d\}\rangle \\ \langle\{a,b,1\},\{a,1,d\}\rangle \end{pmatrix} \\
\Sigma := \min_{\sigma}(\Sigma \oplus \mathrm{prox}(\Sigma\varphi))(\varphi) &= ([a],[c]) \qquad \text{(1.recursion)} \\
\mathrm{prox}(\Sigma\varphi) &= (\{b,c\},\{a,b\}) \\
(\Sigma \oplus \mathrm{prox}(\Sigma\varphi)) &= ([a,b],[a,c],[c,a],[c,b]) \\
(\Sigma \oplus \mathrm{prox}(\Sigma\varphi))(\varphi) &= \begin{pmatrix} \langle\{1,2,c\},\{1,c,d\}\rangle \\ \langle\{1,b,2\},\{1,2,d\}\rangle \\ \langle\{2,b,1\},\{2,1,d\}\rangle \\ \langle\{a,2,1\},\{a,1,d\}\rangle \end{pmatrix} \\
\Sigma := \min_{\sigma}(\Sigma \oplus \mathrm{prox}(\Sigma\varphi))(\varphi) &= ([a,c],[c,a]) \qquad \text{(2.recursion)} \\
&\vdots \\
\Sigma := \min_{\sigma}(\Sigma \oplus \mathrm{prox}(\Sigma\varphi))(\varphi) &= ([a,c,b],[c,a,b]) \qquad \text{(3.recursion)} \\
&\vdots \\
\Sigma := \min_{\sigma}(\Sigma \oplus \mathrm{prox}(\Sigma\varphi))(\varphi) &= ([a,c,b,d],[c,a,b,d]) \qquad \text{(4.recursion)}
\end{aligned}$$

So $([a,c,b,d],[c,a,b,d])$ are exactly those renamings which minimize φ (i.e. our φ_N mentioned at the beginning of this section) namely to $\langle\{1,2,3\},\{1,2,4\}\rangle$. Hence we have two normal forms: $\langle\{a,c,b\},\{a,c,d\}\rangle$ and $\langle\{c,a,b\},\{c,a,d\}\rangle$.

With a slight modification of the ACI normalization algorithm we can also handle AC normalization (i.e. without I). For that we simply have to replace sets in ACI expressions by multisets. Most prominent candidate expressions for AC normalizations are addition and multiplication.

Finally it should be mentioned that AC(I) normalization as introduced here doesn't allow for a more efficient term matching in the worst case than the naive approach does meaning testing all permutations. Such a worst case is an expression where none of its AC(I) subexpressions share common symbols. In practice, however, this is rather the exception as our experiments with the Mizar library shows (s.section 4).

2.3 Concatenating Normalizers to an Overall Normalizer

Now let ∇ be the normalizer obtained by chaining all the normalizers discussed so far in the following order: elimination of $\Rightarrow$ and $\Leftrightarrow$, negation normal form, prenex normal form, and conjunctive normal form, merge binding, ACI normalization, formula abstraction, and sort binding. This order of normalizers is empirically optimal in the sense that each normal form of this overall normalization would represent a maximal equivalence class of formulae; i.e. every other order of normalizers would yield smaller equivalence classes or at most of equal size.

Finally an illustrative example demonstrates the normalization process on some example formula:

	$\forall x.\forall z.R(x,z) \Rightarrow \exists y.(\forall w.Q(y,w) \Rightarrow R(x,y)) \wedge R(y,z))$
$\Rightarrow$-elim	$\forall x.\forall z.\neg R(x,z) \vee \exists y.(\forall w.\neg Q(y,w) \vee R(x,y)) \wedge R(y,z))$
prenex form	$\forall x.\forall z.\exists y.\forall w.\neg R(x,z) \vee ((\neg Q(y,w) \wedge R(x,y)) \vee R(y,z))$
CNF	$\forall x.\forall z.\exists y.\forall w.(\neg R(x,z) \vee \neg Q(y,w) \vee R(x,y)) \wedge (\neg R(x,z) \vee R(y,z))$
merge binding	$\forall x,z.\exists y.\forall w.(\neg R(x,z) \vee \neg Q(y,w) \vee R(x,y)) \wedge (\neg R(x,z) \vee R(y,z))$
∇_{aci}	$\forall x,z.\exists y.\forall w.(R(y,z) \vee \neg R(x,z)) \wedge (R(x,y) \vee \neg R(x,z) \vee \neg Q(y,w))$

The initial and the final formula are equivalent as each normalization step preserves the semantics.

Up to now we have considered equivalence transformations whereby the constants stay the same eventually. For a matching modulo renaming of constants we need a different representation of normal forms gained by the final step which we call **formula abstraction**. This normalization step returns for each input formula φ a pair $(\hat{\varphi}, \overline{p})$ which we call the skeleton and the parameter of φ. The parameter represent the constants of φ in the order from left to right as they occur at first in φ. The skeleton is φ after replacing all its constants by placeholders and a subsequent α-normalization. For instance the formula abstraction of $\forall x.R(x, f(x))$ returns $\forall 1.0(1, 0(1))$ as skeleton and $[R, f]$ as parameter.

Formula abstraction followed by sort binding are finally the last steps of the overall normalizations. In the remainder of this paper we assume a formula being completely normalized through all these steps when we talk of its skeleton and parameter respectively.

3 An Illustrative Example for Knowledge Sharing

Let us now see how the $\nu\epsilon$-retrieval can be used for knowledge sharing and expansion by detection of theory inclusions. The basic idea behind theory inclusion is to find a signature morphism between theories such that the axioms of the translated source theory are theorems in the target theory. For that we normalize all statements of the target theory and all axioms of the source theory. If we can find for each skeleton of the source theory's axioms a syntactically identical statement skeleton from the target theory then we try to find a consistent mapping between the parameter of the source theory and those of the target theory.

If we can find such a mapping between parameters we have found a signature morphism which allows for theory inclusion.

An elementary example from arithmetic should demonstrate how this works in principle. In contrast to the experiments on the MIZAR library reported in section 4, this is only a contrived example aimed at illustrating the process.

For the greatest common divisor (gcd) and the least common multiple (lcm) one finds the properties that both gcd and lcm are associative and commutative. Moreover we have the dual absorption properties:

$$\forall a, b.\, \gcd(a, \mathrm{lcm}(a, b)) = a \qquad \forall a, b.\, \mathrm{lcm}(a, \gcd(a, b)) = a$$

In lattice theory we have two operators $\sqcup$, $\sqcap$ and their associativity and commutativity as axioms as well as absorption. Assume that the absorption axioms are formalized by some author as follows:

$$\forall x. \forall y. x = (y \sqcap x) \sqcup x \qquad \forall x. \forall y. x = (y \sqcup x) \sqcap x$$

All absorption formulae, from arithmetic and from lattice theory, would normalize to a single skeleton $\varphi := \forall 1, 4.1 = 2(1, 3(1, 4)))$ due to the commutativity property of all involved operators. The corresponding parameters are:

$$p_1 = [\gcd, \mathrm{lcm}] \quad p_2 = [\mathrm{lcm}, \gcd] \quad p_3 = [\sqcup, \sqcap] \quad p_4 = [\sqcap, \sqcup]$$

A consistent mapping from the parameters of lattice theory to those of arithmetic is for instance:

$$\sigma := p_1 \circ p_3 = p_2 \circ p_4 = [\sqcap \to \gcd, \sqcup \to \mathrm{lcm}]$$

Thus we translated with σ the axioms of the lattice theory into theorems of arithmetic. The gained knowledge expansion is that all theorems from lattice theory are also theorems in arithmetic after applying this translation. Note that this translation is found via normalization that goes beyond simple α-equivalence, as it also takes the ACI properties of the operators into account.

4 Experiments on a Real World Math Library

MIZAR is a representation format for mathematics that is close to mathematical vernacular used in publications and a deduction system for verifying proofs in the MIZAR language. The continual development of the MIZAR system has resulted a centrally maintained library of mathematics (the MIZAR mathematical library MML [Miz]). The MML is a collection of MIZAR articles: text-files that contain theorems and definitions, and proofs. Currently the MML (version 4.76.959) contains 959 articles with 43149 theorems and 8185 definitions. Introductory information on MIZAR and the MML can be found in [RST01, Wie99]. The MIZAR language is based on Tarski-Grothendiek set theory [Try90], it is essentially a first-order logic with an extremely expressive type systems that features dependent types as well as predicate restrictions, see [Ban03] for details.

In our experiments we don't operate on the MML in its original format, but in its translated equivalent in first order logic format [Urb03], which contains 1530811 formulae distributed over 12529 files. The main reason for using this version is that for our experiment we need a simple notion of theory, namely as a set of axioms (for the purposes of this paper we don't differentiate between definitions and axioms), which constitute the theory, and a set of theorems, which were derived from them. In the original MIZAR format this simple notion is hidden behind combinations of very MIZAR specific notions such as "article", "vocabulary", "notation", "cluster", etc. To map these notions into our simple notion of theory would need a quite deep understanding of the MIZAR system. A structurally faithful translation of the MML into the OMDoc format is currently under way [BK07], and we will rerun our experiments on that and compare the results.

The translated MIZAR library is represented in DFG syntax [HKW96], a first order syntax that was designed to be easily parsed. Its concepts map straightforwardly to our needs: The main object of a file in DFG syntax is a "problem" which we interpret as a "theory". A problem contains arbitrarily many formulae being either of the type "axiom" or "conjecture". What is called conjecture there in DFG syntax corresponds to our theorems as we assume that all conjectures of the MML in DFG syntax are already proven and thus can be called theorems.

4.1 MML and Its Export in DFG-Syntax

We will need to review the basics of Josef Urban's translation of the MML to understand the retrieval experiment: Each object of MML gets a context-independent name and all types and properties (e.g. commutativity, transitivity, etc.) are translated into one or more first order formulae. Formulae are relativized with respect to the typed variables occurring in them. For instance the MIZAR expression

```
for x being Real holds x-x = 0
```

translates to

```
forall([x], implies(v1_arytm(x), equal(k3_real_1(x,x),0))).
```

where `v1_arytm(x)` encodes the type information `x being Real`. The translation approach leads to specific artifacts worth mentioning for our experiments:

- First of all the "one theorem = one (self-contained) problem" principle induces a considerable amount of redundant axiom repetitions in the assumptions.
- The translation from types and type hierarchies to first order formulae causes another blow up of formulae.
- Even worse many of these type translations result in redundant tautologies like `forall([x], implies(and(true,v1_arytm(x)),true))`. Since the basic MIZAR type `set` translates to true all these tautologies result probably from types involving `set`.
- Some formulae, which have less than 20 subterms in the original MML format, transform to monster formulae with over 800 subterms.

To cope with these we had to adapt our initial normalization procedure:

- Normalizing, in particular to CNF, becomes to expensive for large formulae. We decided to exclude formulae with more than 100 subterms from normalization. Such very large formulae wouldn't occur in handwritten math libraries anyway.
- Formulae where an associative and commutative operator have more than 10 arguments are excluded from AC normalization. Again such formulae are rather unlikely in hand written libraries - even after building the CNF.
- The frequent existence of `true` as part of formulae suggested an additional normalization step to eliminate all these `true` occurrences. For instance a formula like `implies(true,and(true,r(x)))` reduces to `r(x)`. Moreover all axioms which reduce to a single `true` are excluded from the insertion into the database since they don't influence the semantic of a theory.
- The `true` elimination normalization step, however, induced the subsequent artifact of con- and disjunctions with only one argument, which were handled the obvious way; e.g. `and(true,r(x))` reduces to `and(r(x))` and finally to `r(x)`

As a statistical result of the database insertion process we found that the original amount of 1530811 formulae can be reduced to 1416653 formulae due to elimination of 114158 tautologies. 18472 of these 1416653 were of that said large size that they were excluded from normalization. With the exclusion of large formula, normalization ran in about two hours on a contemporary PC, which is acceptable for an indexing-time step.

The most interesting number in this phase, however, is the ratio between the number of original formulae vs the number of skeletons. We consider this as a measure of redundancy or the other way round as indicator for potential theory reuse. It turns out that these 1416653 original formulae are instances of just 18155 skeletons; i.e. about every 80 formulae share the same skeleton. It must be said, however, that this factor significantly relies on the "one theorem = one (self-contained) problem" principle (see above). Hence a deeper analysis is needed to find out which percentage of formulae with common skeletons are not just simple copies generated by that principle.

4.2 TheoScrutor: A Knowledge Base Architecture for Normalized Formulae

TheoScrutor, our implementation of a theory search engine, has a simple architecture given in Figure 1. For indexing (i.e. initialization of the database) all files of the MML in DFG syntax are fed to a parser and then normalized and finally all normalized formulae are inserted into a MySQL database. The parser and the normalizer are implemented in Haskell; a database record contains a *file name* and *line number* referencing the occurrence of the normalized formula, its *skeleton*, and its *parameter*.

To query theory inclusions of a source theory S this theory is fed as file in DFG syntax to the request process which consists of the same parse and

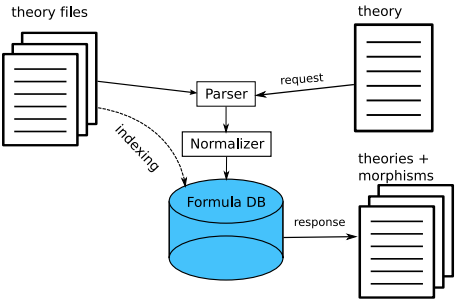

Fig. 1. Architecture of the theory inclusion search engine TheoScrutor

normalization steps as the indexing process. The response of this request is a list of target theories T_i together with a set of signature morphisms $\{\sigma_j\}_{i,j}$. Each pair (i, j) of theory and signature morphism represents a theory inclusions as described in section 3. As additional information to each signature morphism a mapping between source axioms and target axioms or theorems is attached.

4.3 Querying with TheoScrutor

To test the performance of TheoScrutor, we ran various test queries on the database. The first kind of test is a single formula query. This can be considered as special theory inclusion query where the source theory has only one axiom. The test is relevant, because it allows for a performance comparison to the MathWebSearch even though there are differences: MathWebSearch supports subterm instantiation queries (i.e given a term t, it returns all subterms t' of formulae with $\sigma(t') = t$), and we support ν-equivalence. Hence the search result sets can't be the same in general. We are actually comparing ν-retrieval using database indexing technology with term matching using term indexing technology in our test. Moreover the MathWebSearch corpus on which the test runs contains 77000 formulae.

We took as an example query the theorem from the file `aff_1__t40_aff_1` of the DFG syntax version of MML : $\forall.a, b, c.\neg(f(a) \wedge (\neg g(a)(h(a) \wedge l(a)))) \wedge (m(b, u(a)) \wedge m(c, u(a)) \Rightarrow m(r(a, b, c), s(u(a)))$. As query request the theorem is interpreted as axiom. Our search engine needs 200 ms to retrieve from 175 theory inclusions with 88 target theories. Very wide spread formulae are more expensive in search time of course, but still acceptable. As witness query the law of commutativity returns 11502 theory inclusions belonging to 4447 target theories within just 6.4 seconds. An analogous[3] query of commutativity with MathWebSearch took 0.9 seconds returning.

Theory inclusion queries with multiple formulae is actually a distinctive feature of TheoScrutor, though it can partially be simulated by document-scoped Boolean queries in MathWebSearch. Some experiments should give an idea

[3] To simulate the formula abstraction, we replaced symbols with query variables.

of TheoScrutor performance. The first experiment puts focus on a very small source theory whereas the second investigates average size theories: The theory of monoids is constituted by just two axioms: associativity and neutral element. Querying monoids takes 0.2 seconds resulting in 291 theory inclusions with 141 target theories.

For the experiment with average size theories we took theories from MML itself (in DFG syntax of course): the theory `aff_1__t40_aff_1` with 40 axioms. The theory inclusion query took 1.6 seconds finding e.g. different 192 theory inclusions with targets `aff_1__t37_aff_1` and `aff_1__t47_aff_1` (both theories basically share their axioms with the source theory). However, due to the artifact that every theory contains only one theorem, only three different theorem reuses are gained from these 192 theory inclusions.

5 Conclusion and Future Work

We have proposed a normalization-based approach to mathematical formula and theory retrieval modulo an equivalence theory and concept renaming. Concretely, we have developed a waterfall of normalizers that empirically maximize the ϵ-equivalence classes, while keeping normalization tractable during search index creation. One of the strengths of the normalization-based approach is that we can adopt a flexible notion of scope of constant renaming, allowing to tailor the method not only to $\nu\epsilon$-retrieval, but also to theory retrieval. The former is a user-level task for the working mathematician, where constants should keep their meaning, whereas the latter is a knowledge-engineering task for a library maintainer, where constants must be open to renaming for re-interpretation in different contexts.

With TheoScrutor we have an implementation of the proposed approach. We tested it on a real-life task: the Mizar library and shown the steps involved to be tractable (after some practical adaptations).

One may object that the theory inclusions found by our system are relatively trivial from a mathematicians perspective. This is not surprising since normalization is essentially based on pure logical equivalence transformation — sophisticated proofs as mathematicians appreciate are not involved. However, this perspective neglects an important aspect of our original goal, namely to improve the accessibility of knowledge in large digital libraries. Whether a theory inclusion is trivial or not from a mathematicians point of view is secondary if our goal is to expand our knowledge base. Moreover what is folklore to one mathematician in one research area is sometimes completely unknown to another mathematician from a different area and certainly to a mathematically interested layman too.

The strength of automated detection of theory inclusion via normalization is the ability of scanning masses of formulae. Mathematicians are unsurpassable in their dedicated field, but machines are good in precision and mass processing - they can discover useful things which are simply overlooked by humans.

We have concentrated on normalization-based ϵ retrieval in this paper, as the normalization properties of logical connectives and quantifiers are given by the base logic. The normalization could be extended by normalizers for constants that are not abstracted over. To enable this, TheoScrutor would have to scan the source theory axioms for e.g. the statements of the ACI properties of addition. Currently, this is beyond the scope of our implementation.

We will re-run our normalization experiments on the structurally faithful OMDoc translation of the MML currently under way [BK07], and compare the results with the first-order version. It would also be interesting to experiment with other CNF transformations as normalizers, e.g. the very powerful FLOTTER [WGR96] implementation.

References

[Ban03] Bancerek, G.: On the structure of Mizar types. Electronic Notes in Theoretical Computer Science, 85(7) (2003)

[BK07] Bancerek, G., Kohlhase, M.: The mizar mathematical library in omdoc (submitted 2007)

[CAB+86] Robert, L., Constable, S., Allen, H., Bromly, W., Cleaveland, J., Cremer, R., Harper, D., Howe, T., Knoblock, N., Mendler, P., Panangaden, J., Sasaki, J., Smith, S.: Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall, Englewood Cliffs (1986)

[DGH96] Dalmas, S., Gaëtano, M., Huchet, C.: A deductive database for mathematical formulas. In: Limongelli, C., Calmet, J. (eds.) DISCO 1996. LNCS, vol. 1128, pp. 287–296. Springer, Heidelberg (1996)

[HKW96] Hähnle, R., Kerber, M., Weidenbach, C.: Common syntax of dfg-schwerpunktprogramm "deduktion". Interner Bericht 10/96, Universität Karlsruhe, Fakultät für Informatik (1996)

[Kc06] Kohlhase, M., Şucan, I.: A search engine for mathematical formulae. In: Calmet, J., Ida, T., Wang, D. (eds.) AISC 2006. LNCS (LNAI), vol. 4120, pp. 241–253. Springer, Heidelberg (2006)

[Kc07] Kohlhase, M., Şucan, I.: System description: MathWebSearch 0.3, a semantic search engine. submitted to CADE 21 (2007)

[Koh06] Kohlhase, M.: OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]. LNCS (LNAI), vol. 4180. Springer, Heidelberg (2006)

[Miz]

[NP05] Naylor, W., Padget, J.A.: Semantic matching for mathematical services. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 174–189. Springer, Heidelberg (2006)

[Nup] The NuPrl online theory library. Internet interface at `http://simon.cs.cornell.edu/Info/Projects/NuPrl/Nuprl4.2/Libraries/Welc.html`

[ORS92] Owre, S., Rushby, J.M., Shankar, N.: PVS: a prototype verification system. In: Kapur, D. (ed.) Automated Deduction - CADE-11. LNCS, vol. 607, pp. 748–752. Springer, Heidelberg (1992)

[PVS] Pvs libraries. `http://pvs.csl.sri.com/libraries.html`

[RST01] Rudnicki, P., Schwarzweller, C., Trybulec, A.: Commutative algebra in the Mizar system. Journal of Symbolic Computation 32, 143–169 (2001)

[Rud92] Rudnicki, P.: An overview of the mizar project. In: Proceedings of the 1992 Workshop on Types and Proofs as Programs, pp. 311–332 (1992)

[Tea] Coq Development Team. The Coq Proof Assistant Reference Manual. INRIA. see, http://coq.inria.fr/doc/main.html

[Try90] Trybulec, A.: Tarski Grothendieck set theory. Formalized Mathematics 1(1), 9–11 (1990)

[Urb03] Urban, J.: Translating mizar for first-order theorem provers. In: Asperti, A., Buchberger, B., Davenport, J.H. (eds.) MKM 2003. LNCS, vol. 2594, pp. 203–215. Springer, Heidelberg (2003)

[WGR96] Weidenbach, C., Gaede, B., Rock, G.: Spass & flotter, version 0.42. In: McRobbie, M.A., Slaney, J.K. (eds.) Automated Deduction - Cade-13. LNCS, vol. 1104, Springer, Heidelberg (1996)

[Wie99] Wiedijk, F.: Mizar: An impression (1999) http://www.cs.kun.nl/~freek/notes