A content based mathematical search engine: Whelp

Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli

Department of Computer Science, University of Bologna
Mura Anteo Zamboni, 7 — 40127 Bologna, ITALY
{asperti,fguidi,sacerdot,tassi,zacchiro}@cs.unibo.it

Abstract. The prototype of a content based search engine for mathematical knowledge supporting a small set of queries requiring matching and/or typing operations is described. The prototype — called Whelp — exploits a metadata approach for indexing the information that looks far more flexible than traditional indexing techniques for structured expressions like substitution, discrimination, or context trees. The prototype has been instantiated to the standard library of the Coq proof assistant extended with many user contributions.

1 Introduction

The paper describes the prototype of a content based search engine for mathematical knowledge — called Whelp — developed inside the European Project IST-2001-33562 MoWGLI [4]. Whelp has been mostly tested to search notions inside the library of formal mathematical knowledge of the Coq proof assistant [8]. Due to its dimension (about 40,000 theorems), this library was adopted by MoWGLI as a main example of repository of structured mathematical information. However, Whelp — better, its filtering phase — only works on a small set of metadata automatically extracted from the structured sources, and is thus largely independent from the actual syntax (and semantics) of the information. Metadata also offer a higher flexibility with respect to more canonical indexing techniques such as discrimination trees [17], substitution trees [14] or context trees [13] since all these approaches are optimized for the single operation of (forward) matching, and are difficult to adapt or tune with additional constraints (such as global constraints on the signature of the term, just to make a simple but significant example).

Whelp is the final output of a three-year research work inside MoWGLI which consisted in exporting the Coq library into XML, defining a suitable set of metadata for indexing the information, implementing the actual indexing tools, and finally designing and developing the search engine. Whelp itself is the result of a complete architectural re-visitation of a first prototype described in [15], integrated with the efficient retrieval mechanisms described in [3] (further improved as described in Sect. 5.2), and integrated with syntactic facilities borrowed from the disambiguating parser of [20]. Since the prototype version described in [15], also the Web interface has been completely rewritten and simplified, exploiting most of the publishing techniques developed for the hypertextual rendering of the Coq library (see `http://helm.cs.unibo.it/`) and described in Sect. 6.

2 Syntax

Whelp interacts with the user as a classical World Wide Web search engine, it expects single line queries and returns a list of results matching it. Whelp currently supports four different kinds of queries, addressing different user-requirements emerged in MoWGLI: MATCH, HINT, ELIM, and LOCATE (described in Sect. 5). The list is not meant to be exhaustive and is likely to grow in the future.

The most typical of these queries (MATCH and HINT) require the user to input a term of the Calculus of (co-)Inductive Constructions — CIC — (the underlying calculus of Coq), supporting different kinds of pattern based queries. Nevertheless, the concrete syntax we chose for writing the input term is not bound to any specific logical system: it has been designed to be as similar as possible to ordinary mathematics formulae, in their TEX encoding (see Table 1).

term	::=	*identifier*	
	|	*number*	
	|	`Prop` | `Type` | `Set`	sort
	|	`?`	placeholder
	|	*term term*	application
	|	*binder vars* `.` *term*	abstraction
	|	*term* `\to` *term*	arrow type
	|	`(` *term* `)`	grouping
	|	*term binop term*	binary operator
	|	*unop term*	unary operator
binder	::=	`\forall` | `\exists` | `\lambda`	
vars	::=	*names*	variables
	|	*names* `:` *term*	typed variables
names	::=	*identifier* | *identifier names*	
binop	::=	`+` | `-` | `*` | `/` | `^`	arithmetic operators
	|	`<` | `>` | `\leq` | `\geq` | `=` | `\neq`	comparison operators
	|	`\lor` | `\land`	logical operators
unop	::=	`-`	unary minus
	|	`\lnot`	logical negation

Table 1. Whelp's term syntax

As a consequence of the generality of syntax, user provided terms do not usually have a unique direct mapping to CIC term, but must be suitably interpreted in order to solve *ambiguities.* Consider for example the following term input: `\forall x. 1*x = x`. in order to find the corresponding CIC term we need to know the possible meanings of the number `1` and the symbol `*`.[1]

The typical processing of a user query (depicted in Fig. 1) is therefore a pipeline made of four distinct phases: *parsing* (canonical transformation from concrete textual syntax to Abstract Syntax Trees, ASTs for short), *disambiguation* (described in next section), *metadata extraction* (described in Sect. 4), and the actual *query* (described in Sect. 5).

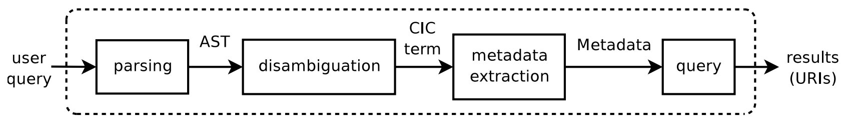


Fig. 1. Whelp's processing

3 Disambiguation

The disambiguation phase builds CIC terms from ASTs of user inputs (also called *ambiguous terms*). Ambiguous terms may carry three different *sources of ambiguity*: unbound identifiers, literal numbers, and literal symbols. *Unbound identifiers* are sources of ambiguity since the same name could have been used to represent different objects. For example, three different theorems of the Coq library share the name *plus_assoc* (locating them is an exercise for the interested reader. Hint: use Whelp's LOCATE query).

Numbers are ambiguous since several different encodings of them could be provided in logical systems. In the Coq standard library for example we found naturals (in their unary encoding), positives (binary encoding), integers (signed positives), and reals. Finally, *symbols* (instances of the *binop* and *unop* syntactic categories of Table 1) are ambiguous as well: infix `+` for example is overloaded to represent addition over the four different kinds of numbers available in the Coq standard library. Note that given a term with more than one sources of ambiguity, not all possible disambiguation choices are valid: for example, given the input `1+1` we must choose an interpretation of `+` which is typable in CIC according to the chosen interpretation for `1`; choosing as `+` the addition over natural numbers and as `1` the real number 1 will lead to a type error.

A *disambiguation algorithm* takes as input an ambiguous term and return a fully determined CIC term. The *naive disambiguation algorithm* takes as input an ambiguous term t and proceeds as follows:

1. Create disambiguation domains $\{D_i | i \in Dom(t)\}$, where $Dom(t)$ is the set of ambiguity sources of t. Each D_i is a set of CIC terms.

[1] Note that `x` is not undetermined, since it is a bound variable.

2. Let $\Phi = \{\phi_i | i \in Dom(t), \phi_i \in D_i\}$ be an interpretation for t. Given t and an interpretation Φ, a CIC term is fully determined. Iterate over all possible interpretations of t and type-check them, keep only typable interpretations (i.e. interpretations that determine typable terms).
3. Let n be the number of interpretations who survived step 2. If $n = 0$ signal a type error. If $n = 1$ we have found exactly one CIC term corresponding to t, returns it as output of the disambiguation phase. If $n > 1$ let the user choose one of the n interpretations and returns the corresponding term.

The above algorithm is highly inefficient since the number of possible interpretations Φ grows exponentially with the number of ambiguity sources. The actual algorithm used in Whelp is far more efficient being, in the average case, linear in the number of ambiguity sources.

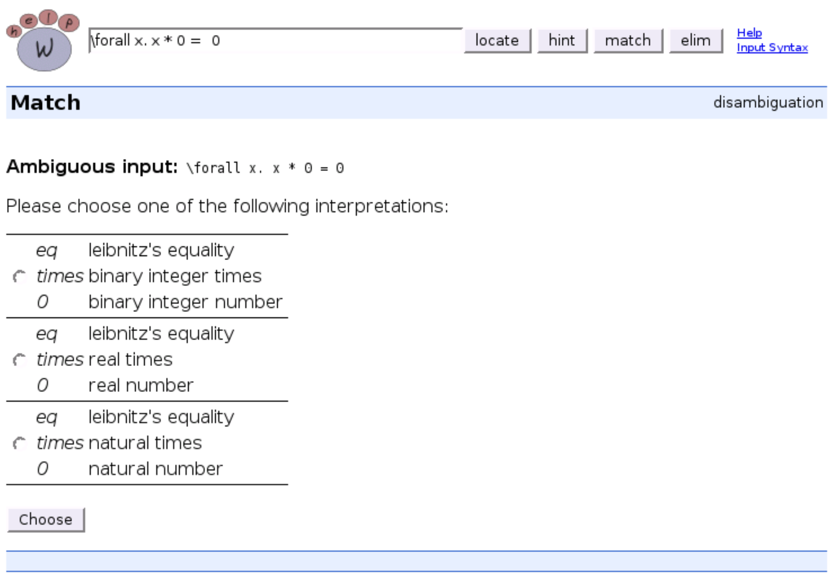
The efficient algorithm can be applied if the logic can be extended with metavariables and a refiner can be implemented. This is the case for CIC and several other logics. *Metavariables* [18] are typed, non linear placeholders that can occur in terms; $?_i$ usually denotes the i-th metavariable, while ? denotes a freshly created metavariable. A *refiner* [16] is a function whose input is a term with placeholders and whose output is either a new term obtained instantiating some placeholder or ϵ, meaning that no well typed instantiation could be found for the placeholders occurring in the term (type error).

The efficient algorithm starts with an interpretation $\Phi_0 = \{\phi_i | \phi_i = ?, i \in Dom(t)\}$, which associates a fresh metavariable to each source of ambiguity. Then it iterates refining the current CIC term (i.e. the term obtained interpreting t with Φ_i). If the refinement succeeds the next interpretation Φ_{i+1} will be created *making a choice*, that is replacing a placeholder with one of the possible choice from the corresponding disambiguation domain. The placeholder to be replaced is chosen following a preorder visit of the ambiguous term. If the refinement fails the current set of choices cannot lead to a well-typed term and backtracking is attempted. Once an unambiguous correct interpretation is found (i.e. Φ_i does no longer contain any placeholder), backtracking is attempted anyway to find the other correct interpretations.

The intuition which explain why this algorithm is more efficient is that as soon as a term containing placeholders is not typable, no further instantiation of its placeholders could lead to a typable term. For example, during the disambiguation of user input `\forall x. x*0 = 0`, an interpretation Φ_i is encountered which associates ? to the instance of `0` on the right, the real number 0 to the instance of `0` on the left, and the multiplication over natural numbers (`mult` for short) to `*`. The refiner will fail, since `mult` require a natural argument, and no further instantiation of the placeholder will be tried.

If, at the end of the disambiguation, more than one possible interpretations are possible, the user will be asked to choose the intended one (see Fig. 2).

Details of the disambiguation algorithm of Whelp can be found in [20], where an equivalent algorithm that avoids backtracking is also presented.



Match disambiguation

Ambiguous input: `\forall x. x * 0 = 0`

Please choose one of the following interpretations:

	eq	leibnitz's equality
○	*times*	binary integer times
	0	binary integer number
	eq	leibnitz's equality
○	*times*	real times
	0	real number
	eq	leibnitz's equality
○	*times*	natural times
	0	natural number

Choose

Fig. 2. Disambiguation: interpretation choice

4 Metadata

The metadata model we use for indexing mathematical notions is essentially based on a single ternary relation $Ref_p(s,t)$ stating that an object s refers an object t at a given position p. We use a minimal set of positions discriminating the hypotheses (H), from the conclusion (C) and the proof (P) of a theorem (respectively, the type of the input parameters, the type of the result, and the body of a definition). Moreover, in the hypothesis and in the conclusion we also distinguish the root position (MH and MC, respectively) from deeper positions (that, in a first order setting, essentially amounts to distinguish relational symbols from functional ones). Extending the set of positions we could improve the granularity and the precision of our indexing technique but so far, apart from a simple extension discussed below, we never felt this need.

Example 1. Consider the statement:

$$\forall m, n : nat.m \leq n \to m < (S\ n)$$

its metadata are described by the following table:

Symbol	Position
nat	MH
$\leq$	MH
$<$	MC
S	C

All occurrences of bound variables in position MH or MC are collapsed under a unique reserved name *Rel*, forgetting the actual variable name. The occurrences in other positions are not considered. See Sect. 5.3 for an example of use.

The accuracy of metadata for discriminating the statements of the library is remarkable. We computed (see [1]) that the average number of mathematical notions in the Coq library sharing the same metadata set is close to the actual number of *duplicates* (i.e. metadata almost precisely identify statements).

According to the type as proposition analogy, the metadata above may be also used to index the type of functions. For instance, functions from *nat* to *R* (reals) would be identified by the following metadata:

Symbol	Position
nat	MH
R	MC

in this case, however, the metadata model is a bit too rough, since for instance functions of type $nat \to nat$, $nat \to nat \to nat$, $(nat \to nat) \to nat \to nat$ and so on would all share the following metadata set:

Symbol	Position
nat	MH
nat	MC

To improve this situation, we add an integer to MC (MH), expressing the number of parameters of the term (respectively, of the given hypothesis). We call *depth* this value since in the case of CIC is equal to the nesting depth of dependent products[2] along the spine relative to the given position. For instance, the three types above would now have the following metadata sets:

$nat \to nat$

Symbol	Position
nat	MH(0)
nat	MC(1)

$nat \to nat \to nat$

Symbol	Position
nat	MH(0)
nat	MC(2)

$(nat \to nat) \to nat \to nat$

Symbol	Position
nat	MH(1)
nat	H
nat	MH(0)
nat	MC(2)

The depth is a technical improvement that is particularly important for retrieving functions from their types (we shall also see a use in the Elim query, Sect. 5.3), but is otherwise of minor relevance. In the following examples, we shall always list it among the metadata for the sake of completeness, but it may be usually neglected by the reader.

[2] Recall that in type theory, the function space is just a degenerate case of dependent product.

5 Whelp queries

5.1 Match

Not all mathematical results have a canonical name or a set of keywords which could easily identify them. For this reason, it is extremely useful to be able to search the library by means of the explicit statement. More generally, exploiting the well-known types-as-formulae analogy of Curry-Howard, Whelp's MATCH operation takes as input a type and returns a list of objects (definition or proofs) inhabiting it.

Example 2. Find a proof of the distributivity of times over plus on natural numbers. In order to retrieve those statements, Whelp need to be fed with the distributivity law as input: `\forall x,y,z:nat. x * (y+z) = x*y + x*z`. The MATCH query will return 4 results:

1. `cic:/Coq/Arith/Mult/mult_plus_distr_l.con`
2. `cic:/Coq/Arith/Mult/mult_plus_distr_r.con`
3. `cic:/Rocq/SUBST/comparith/mult_plus_distr_r.con`
4. `cic:/Sophia-Antipolis/HARDWARE/GENE/Arith_compl/mult_plus_distr2.con`

(1), (3), and (4) have types which are α-convertible with the user query; (2) is an interesting "false match" returned by Whelp having type $\forall n, m, p \in \mathbb{N}.(n+m)*p = n*p+m*p$, i.e. it is the symmetric version of the distributivity proposition we were looking for.

The match operation simply amounts to revert the indexing operation, looking for terms matching the metadata set computed from the input. For instance, the term `\forall x,y,z:nat. x * (y+z) = x*y + x*z` has the following metadata:

Symbol	Position
nat	MH(0)
$=$	MC(3)
nat	C
$*$	C
$+$	C

Note that *nat* occurs in conclusion as an hidden parameter of equality; the indexed term is the term after disambiguation, not the user input.

Searching for the distributivity law then amounts to look for a term s such that :

$$Ref_{MH(0)}(s, nat) \wedge Ref_{MC(3)}(s, =) \wedge Ref_{C}(s, nat) \wedge Ref_{C}(s, *) \wedge Ref_{C}(s, +)$$

In a relational database, this is a simple and efficient join operation.

Example 3. Suppose we are interested in looking for a definition of summation for series of natural numbers. The type of such an object is something of the kind $(nat \to nat) \to nat \to nat \to nat$, taking the series, two natural numbers expressing summation lower and upper bound, and giving back the resulting sum. Feeding Whelp's MATCH query with such a type does give back four results:

1. `cic:/Coq/Reals/Rfunctions/sum_nat_f.con`
2. `cic:/Sophia-Antipolis/Bertrand/Product/prod_nm.con`
3. `cic:/Sophia-Antipolis/Bertrand/Summation/sum_nm.con`
4. `cic:/Sophia-Antipolis/Rsa/Binomials/sum_nm.con`

Although we have a definition for summation in the standard library, namely *sum_nat_f*, its theory is very underdeveloped. Luckily we have a much more complete development for *sum_nm* in a contribution from Sophia, where:

$$\mathit{sum\_nm}\ n\ m\ f = \sum_{x=n}^{n+m} f(x)$$

Having discovered the name for summation, we may then inquire about the proofs of some of its properties; for instance, considering the semantics of *sum_nm*, we may wonder if the following statement is already in the library:

$$\forall m, n, c : \mathit{nat}.(\mathit{sum\_nm}\ n\ m\ \lambda x : \mathit{nat}.c) = (S\ m) * c$$

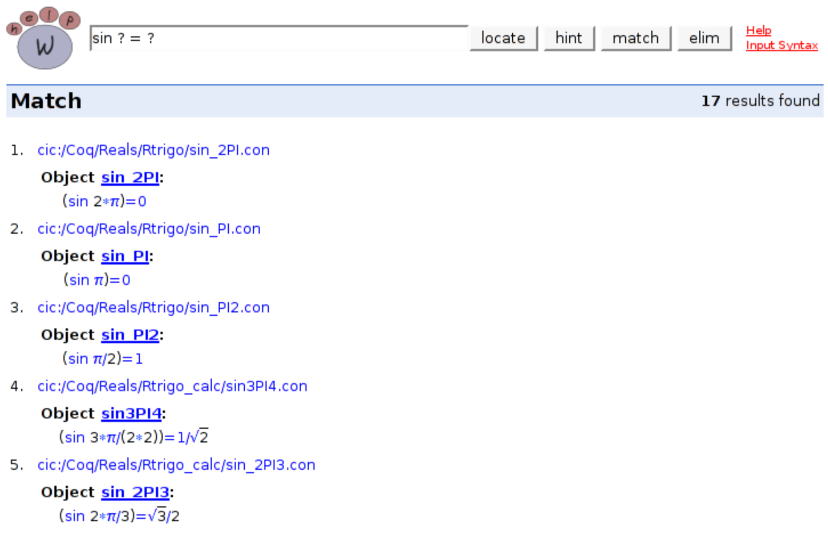
Matching the previous theorem actually succeed, returning the following:
`cic:/Sophia-Antipolis/Bertrand/Summation/sum_nm_c.con`.

Matching incomplete patterns Whelp also support matching with partial patterns, i.e. patterns with placeholders denoted by ?. The approach is essentially identical to the previous one: we compute all the constants A appearing in the pattern, and look for all terms referring at least the set of constants in A, at suitable positions.

Suppose for instance that you are interested in looking for all known facts about the computation of *sin* on given reals. You may just ask Whelp to MATCH *sin* ? = ?, that would result in the following list (plus a couple of spurious results due to the fact that Coq variables are not indexed, at present):

1. `cic:/Coq/Reals/Rtrigo/sin_2PI.con`
2. `cic:/Coq/Reals/Rtrigo/sin_PI.con`
3. `cic:/Coq/Reals/Rtrigo/sin_PI2.con`
4. `cic:/Coq/Reals/Rtrigo_calc/sin3PI4.con`
5. `cic:/Coq/Reals/Rtrigo_calc/sin_2PI3.con`
6. `cic:/Coq/Reals/Rtrigo_calc/sin_3PI2.con`
7. `cic:/Coq/Reals/Rtrigo_calc/sin_5PI4.con`
8. `cic:/Coq/Reals/Rtrigo_calc/sin_PI3.con`
9. `cic:/Coq/Reals/Rtrigo_calc/sin_PI3_cos_PI6.con`
10. `cic:/Coq/Reals/Rtrigo_calc/sin_PI4.con`
11. `cic:/Coq/Reals/Rtrigo_calc/sin_PI6.con`
12. `cic:/Coq/Reals/Rtrigo_calc/sin_PI6_cos_PI3.con`
13. `cic:/Coq/Reals/Rtrigo_calc/sin_cos5PI4.con`
14. `cic:/Coq/Reals/Rtrigo_calc/sin_cos_PI4.con`
15. `cic:/Coq/Reals/Rtrigo_def/sin_0.con`

Previous statements semantics is reasonably clear by their names; Whelp, however, also performs in-line expansion of the statements, and provides hyperlinks to the corresponding proofs (see Fig. 3).



Match **17** results found

1. cic:/Coq/Reals/Rtrigo/sin_2PI.con
   Object **sin_2PI**:
   (sin 2∗π)=0
2. cic:/Coq/Reals/Rtrigo/sin_PI.con
   Object **sin_PI**:
   (sin π)=0
3. cic:/Coq/Reals/Rtrigo/sin_PI2.con
   Object **sin_PI2**:
   (sin π/2)=1
4. cic:/Coq/Reals/Rtrigo_calc/sin3PI4.con
   Object **sin3PI4**:
   (sin 3∗π/(2∗2))=1/√2
5. cic:/Coq/Reals/Rtrigo_calc/sin_2PI3.con
   Object **sin_2PI3**:
   (sin 2∗π/3)=√3/2

Fig. 3. MATCH results.

5.2 Hint

In a process of backward construction of a proof, typical of proof assistants, one is often interested in knowing what theorems can be applied to derive the current goal. The HINT operation of Whelp is exactly meant to this purpose.

Formally, given a goal g and a theorem $t_1 \to t_2 \to \cdots \to t_n \to t$, the problem consists in checking if there exist a substitution θ such that:

$$t\theta = g \qquad (1)$$

A necessary condition for (1), which provides a very efficient filtering of the solution space, is that the set of constants in t must be a subset of those in g. In terms of our metadata model, the problem consists to find all s such as

$$\{x|Ref(s,x)\} \subseteq A \qquad (2)$$

where A is the set of constants in g. This is not a simple operation: the naive approach would require to iterate, for every possible source s, the computation of its forward references, i.e. of $\{x|Ref(s,x)\}$, followed by a set comparison with A.

The solution of [3] is based on the following remarks. Let us call $Card(s)$ the cardinality of $\{x|Ref(s,x)\}$, which can be pre-computed for every s. Then, (2) holds if and only if there is a subset A' of A such that $A' = \{x|Ref(s,x)\}$, or equivalently:

$$A' \subseteq \{x|Ref(s,x)\} \wedge |A'| = Card(s)$$

and finally:

$$\bigwedge_{a \in A'} Ref(s,a) \land |A'| = Card(s)$$

The last one is a simple join that can be efficiently computed by any relational database. So the actual cost is essentially bounded by the computation of all subsets of A, and A, being the signature of a formula, is never very large (and often quite small).

The problem of matching against a large library of heterogeneous notions is however different and far more complex than in a traditional theorem proving setting, where one typically works with respect to a given theory with a fixed, and usually small signature. If e.g. we look for theorems whose conclusion matches some kind of equation like $e_1 = e_2$ we shall eventually find in the library *a lot* of injectivity results relative to operators we are surely not interested in: in a library of 40,000 theorems like the one of Coq we would get back about 3,000 of such *silly matches*. Stated in other words, canonical indexing techniques specifically tailored on the matching problem such as discrimination trees [17], substitution trees [14] or context trees [13] are eventually doomed to fail in a mathematical knowledge management context where one cannot assume a preliminary knowledge on term signatures.

On the other side, the metadata approach is much more flexible, allowing a simple integration of matching operation with additional and different constraints. For instance in the version of *hint* described in [15] the problem of reducing the number of *silly matches* was solved by requiring at least a minimal intersection between the signatures of the two matching terms. However, this approach did sometimes rule out some interesting answers. In the current version the problem has been solved imposing further constraints of the full signature of the term (in particular on the hypothesis), essentially filtering out all solutions that would extend the signature of the goal. The actual implementation of this approach requires a more or less trivial extension to hypothesis of the methodology described in [3].

5.3 Elim

Most statements in the Coq knowledge base concern properties of functions and relations over algebraic types. Proofs of such statements are carried out by structural induction over these types. In particular, to prove a goal by induction over the type t, one needs to apply a lemma stating an induction principle over t (an *eliminator* of t [16]) to that goal. Since many different eliminators can be provided for the same type t (either automatically generated from t, or set up by the user), it is convenient to have a way of retrieving all the eliminators of a given type. The ELIM query of Whelp does this job. To understand how it works, let's take the case of the algebraic type of the natural numbers: one feeds Whelp with the identifier *nat*, which denotes this type in the knowledge base, and expects to find at least the well-known induction principle *nat_ind*:

$$\forall P : nat \to Prop.(P\ 0) \to (\forall n : nat.(P\ n) \to (P\ (S\ n))) \to \forall n : nat.(P\ n)$$

A fairly good approximation of this statement is based on the following observations: the first premise (*nat* $\to$ *Prop*) has an antecedent, a reference to *Prop* in its root and a reference to *nat*; the forth premise has no antecedents and a reference to *nat* in its root; the conclusion contains a bound variable in its root (i.e. P). Notice that we choose not to approximate the major premises of *nat_ind* (the second and the third) because they depend on the structure of *nat* and discriminate the different induction principles over this type.

Thus, a set of constraints approximating *nat_ind* is the following (recall that *Rel* stands for an arbitrary bound variable):

Symbol	Position
Prop	MH(1)
nat	H
nat	MH(0)
Rel	MC

The Elim query of Whelp simply generalizes this scheme substituting *nat* for a given type t and retrieving any statement c such that:

$$Ref_{MH(1)}(c, Prop) \wedge Ref_{H}(c, t) \wedge Ref_{MH(0)}(c, t) \wedge Ref_{MC}(c, Rel)$$

In the case of *nat*, Elim returns 47 statements ordered by the frequency of their use in the library (as expected *nat_ind* is the first one).

5.4 Locate

Whelp's Locate query implements a simple "lookup by name" for library notions. Once fed with an identifier i, Locate returns the list of all objects whose name is i. Intuitively, the *name* of an object contained in library is the name chosen for it by its author.[3]

This list is obtained querying the specific relational metadata $Name(c, i)$ that binds each unit of knowledge c to an identifier i (its *name*). Unix-shell-like wildcards can be used to specify an incomplete identifier: all objects whose name matches the incomplete identifier are returned.

Even if not based on the metadata model described in Sect. 4, Locate turns out to be really useful to browse the library since quite often one remembers the name of an object, but not the corresponding contribution.

Example 4. By entering the name *gcd*, Locate returns four different versions of the "greatest common divisor":

1. `cic:/Orsay/Maths/gcd/gcd.ind#xpointer(1/1)`
2. `cic:/Eindhoven/POCKLINGTON/gcd/gcd.con`
3. `cic:/Sophia-Antipolis/Bertrand/Gcd/gcd.con`
4. `cic:/Sophia-Antipolis/Rsa/Divides/gcd.con`

[3] In the current implementation object names correspond to the last fragment of object URIs, without extension.

6 Web interface

The result of Whelp is, for all queries, a list of URIs (unique identifiers) for notions in the library. This list is not particularly informative for the user, who would like to have hyperlinks or, even better, in-line expansion of the notions.

In the MoWGLI project we developed a service to render on the fly, via a complex chain of XSLT transformations, mathematical objects encoded in XML (those objects of the library of Coq that Whelp is indexing). XSLT is the standard presentational language for XML documents and an XSLT transformation (or *stylesheet*) is a purely functional program written in XSLT that describes a simple transformation between two XML formats.

The service can also render *views* (misleadingly called "theories" in [2]), that is an arbitrary, structured collection of mathematical objects, suitably assembled (by an author or some mechanical tool) for presentational purposes. In a view, definitions, theorems, and so on may be intermixed with explanatory text or figures, and statements are expanded without proofs: a link to the corresponding proof objects allows the user to inspect proofs, if desired.

Providing Whelp with an appealing user interface for presenting the answers (see Fig. 3) has been as simple as making it generate a view and pipelining Whelp with UWOBO,[4] the component of our architecture that implements the rendering service.

UWOBO is a stylesheet manager implemented in OCaml[5] and based on LibXSLT, whose main functionality is the application of a list of stylesheets (each one with the respective list of parameters) to a document. The stylesheets are pre-compiled to improve performance. Both stylesheets and the document are identified using HTTP URLs and can reside on any host. UWOBO is both a Web server and a Web client, accepting processing requests and asking for the document to be processed. Whelp is a Web server, accepting queries as processing requests and returning views to the client.

The Whelp interface is thus simply organized as a HTTP pipeline (see Fig. 4).

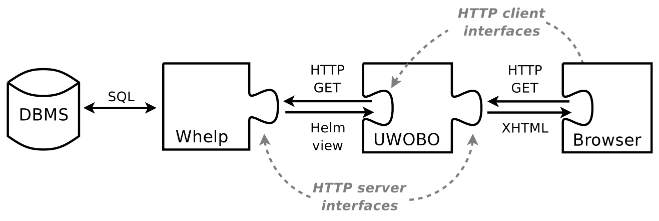


Fig. 4. Whelp's HTTP pipeline.

[4] `http://helm.cs.unibo.it/software/uwobo/`
[5] `http://caml.inria.fr/`

7 Conclusions

Whelp is the Web searching helper developed at the University of Bologna as a part of the European Project IST-2001-33562 MoWGLI. HELP is also the acronym of the four operations currently supported by the system: Hint, Elim, Locate and Pattern-matching.

Much work remains to be done, spanning from relatively simple technical improvements, to more complex architectural re-visitations concerning the indexing technique and the design and implementation of the queries.

Among the main technical improvements which we plan to support in a near future there are:

1. the possibility to confine the search to sub-libraries, and in particular to the standard library alone (this is easy due to the paths of names);
2. skipping the annoying dialog phase with the user during disambiguation for the choice of the intended interpretation, replacing it with a direct investigation of all possibilities;
3. interactive support for advanced queries, allowing the user to directly manipulate the metadata constraints (very powerful, if properly used).

The current indexing politics has some evident limitations, resulting in unexpected results of queries.

The most annoying problem is due to the current management of Coq variables. Roughly, in Coq, *variables* are meant for *declarations*, while *constants* are meant for *definitions*. The current XML-exportation module of Coq [19] does not discharge section variables, replacing this operation with an explicit substitution mechanism; in particular, variables may be instantiated and their status look more similar to local variables than to constants. For this reason, variables have not been indexed; that currently looks as a mistake.

A second problem is due to coercions. The lack of an explicit mechanism for composition of coercions tends to clutter the terms with long chains of coercions, which in case of big algebraic developments as e.g. C-Corn [9], can easily reach about ten elements. The fact that an Object r refers a coercion c contains very little information, especially if coercions typically come in a row, as in Coq. In the future, we plan to skip coercions during indexing.

The final set of improvements concerns the queries. A major issue, for all kinds of content based operations, is to take care, at some extent, of delta reduction. For instance, in Coq, the elementary order relations over natural numbers are defined in terms of the *less or equal* relation, that is a suitable inductive type. Every query concerning such relations could be thus reduced to a similar one about the *less or equal* relation by delta reduction. Even more appealing it looks the possibility to extend the queries up to equational rewriting of the input (via equations available in the library).[6]

[6] The possibility of considering search up to isomorphism (see [10,11]) looks instead less interesting, because our indexing policy is an interesting surrogate that works very well in practice while being much simpler than search up to isomorphisms.

Similarly, the HINT operation, could and should be improved by suitably tuning the current politics for computing the *intended* signature of the search space (for instance, "closing" it by delta reduction or rewriting, adding constructors of inductive types, and so on).

Different kind of queries could be designed as well. An obvious generalization of HINT is a AUTO, automatically attempting to solve a goal by repeated applications of theorems of the library (a deeper exploration of the search space could be also useful for a better rating of the HINT results).

A more interesting example of content-based query that exploits the higher order nature of the input syntax is to look for all mathematical objects providing examples, or instances, of a given notion. For instance we may define the following property, asserting the commutativity of a generic function f

$$is\_commutative := \lambda A : Set.\lambda f : A \to A \to A.\forall x, y : A.(f\ x\ y) = (f\ y\ x)$$

Then, an INSTANCE query should be able to retrieve from the library all commutative operations which have been defined.[7]

To conclude, we remark that the only component of Whelp that is dependent on CIC, the logic of the Coq system, is the disambiguator for the user input. Moreover, the algorithm used for disambiguation depends only on the existence of a refiner for mathematical formulae extended with placeholders. Thus Whelp can be easily modified to index and search other mathematical libraries provided that the statements of the theorems can be easily parsed (to extract the metadata) and that there exists a service to render the results of the queries. In particular, the Mizar development team has just released version 7.3.01 of the Mizar proof assistant that provides both a native XML format and XSLT stylesheets to render the proofs. Thus it is now possible to instantiate Whelp to work on the library of Mizar, soon making possible a direct comparison on the field between Whelp and MML Query, the new search engine for Mizar articles described in [6].

References

1. A. Asperti, F. Guidi, L. Padovani, C. Sacerdoti Coen, I. Schena. *The Science of Equality: some statistical considerations on the Coq library.* Mathematical Knowledge Management Symposium, 25-29 November 2003, Heriot-Watt University, Edinburgh, Scotland.
2. A. Asperti, F. Guidi, L. Padovani, C. Sacerdoti Coen, I. Schena. *Mathematical Knowledge Management in HELM.* Annals of Mathematics and Artificial Intelligence, 38(1): 27–46; May 2003.
3. A. Asperti, M. Selmi. *Efficient Retrieval of Mathematical Statements.* In Proceeding of the Third International Conference on Mathematical Knowledge Management, MKM 2004. Bialowieza, Poland. LNCS 3119.

[7] MATCH is in fact a particular case of INSTANCE where the initial sequence of lambda abstractions is empty.

4. A.Asperti, B.Wegner. *An Approach to Machine-Understandable Representation of the Mathematical Information in Digital Documents.* In: Fengshai Bai and Bernd Wegner (eds.): Electronic Information and Communication in Mathematics, LNCS vol. 2730, pp. 14–23, 2003
5. G. Bancerek, P. Rudnicki. *Information Retrieval in MML.* In A.Asperti, B.Buchberger,J.Davenport (eds), Proceedings of the Second International Conference on Mathematical Knowledge Management, MKM 2003. LNCS, 2594.
6. G. Bancerek, J.Urban. *Integrated Semantic Browsing of the Mizar Mathematical Repository.* In: A.Asperti, G.Bancerek, A.Trybulec (eds.), Proceeding of the Third International Conference on Mathematical Knowledge Management, Springer LNCS 3119.
7. P.Cairns. *Informalising Formal Mathematics: Searching the Mizar Library with Latent Semantics.* In Proceeding of the Third International Conference on Mathematical Knowledge Management, MKM 2004. Bialowieza, Poland. LNCS 3119.
8. The Coq proof-assistant, `http://coq.inria.fr`
9. L. Cruz-Filipe, H. Geuvers, F. Wiedijk, "C-CoRN, the Constructive Coq Repository at Nijmegen". In: A.Asperti, G.Bancerek, A.Trybulec (eds.), Proceeding of the Third International Conference on Mathematical Knowledge Management, Springer LNCS 3119, 88-103, 2004.
10. D.Delahaye, R. Di Cosmo. Information Retrieval in a Coq Proof Library using Type Isomorphisms. In Proceedings of TYPES 99, Lökeberg. Springer-Verlag LNCS, 1999.
11. R. Di Cosmo. *Isomorphisms of Types: from Lambda Calculus to Information Retrieval and Language Design*, Birkhauser, 1995, IBSN-0-8176-3763-X.
12. D. Draheim, W. Neun, D. Suliman. *Classifying Differential Equations on the Web.* In Proceeding of the Third International Conference on Mathematical Knowledge Management, MKM 2004. LNCS, 3119.
13. H. Ganzinger, R. Nieuwehuis, P. Nivela. *Fast Term Indexing with Coded Context Trees. Journal of Automated Reasoning.* To appear.
14. P.Graf. Substitution Tree Indexing. Proceedings of the 6th RTA Conference, Springer-Verlag LNCS 914, pp. 117-131, Kaiserlautern, Germany, April 4-7, 1995.
15. F. Guidi, C. Sacerdoti Coen. *Querying Distributed Digital Libraries of Mathematics.* In Proceedings of Calculemus 2003, 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning. Aracne Editrice.
16. C. McBride. Dependently Typed Functional Programs and their Proofs. Ph.D. thesis, University of Edinburgh, 1999.
17. W. McCune. *Experiments with discrimination tree indexing and path indexing for term retrieval.* Journal of Automated Reasoning, 9(2):147-167, October 1992.
18. C. Munoz. A Calculus of Substitutions for Incomplete-Proof Representation in Type Theory. Ph.D. thesis, INRIA, 1997.
19. C. Sacerdoti Coen. *From Proof-Assistans to Distributed Libraries of Mathematics: Tips and Pitfalls.* In Proc. Mathematical Knowledge Management 2003, Lecture Notes in Computer Science, Vol. 2594, pp. 30–44, Springer-Verlag.
20. C. Sacerdoti Coen, S. Zacchiroli. *Efficient Ambiguous Parsing of Mathematical Formulae.* In Proceedings of the Third International Conference on Mathematical Knowledge Management, MKM 2004. LNCS, 3119.