Querying Distributed Digital Libraries of Mathematics

Ferruccio Guidi, Claudio Sacerdoti Coen
{fguidi,sacerdot}@cs.unibo.it
Department of Computer Science,
University of Bologna

Rome, September 10, 2003

The Goal

We need to query mathematical digital libraries to retrieve definitions, axioms and proved statements.

The Goal

We need to query mathematical digital libraries to retrieve definitions, axioms and proved statements.

- Query-by-name : totally unreliable and not useful ; very quick

The Goal

We need to query mathematical digital libraries to retrieve definitions, axioms and proved statements.

- Query-by-name : totally unreliable and not useful ; very quick
- Query-by-keyword+dc : it used to be the only practical way those old times when documents where unstructured ; not precise, but very quick

The Goal

We need to query mathematical digital libraries to retrieve definitions, axioms and proved statements.

- Query-by-name : totally unreliable and not useful ; very quick
- Query-by-keyword+dc : it used to be the only practical way those old times when documents where unstructured ; not precise, but very quick
- Query-by-matching : it works for formal mathematics and for structured mathematics (content level) ; super-linear in the size of the library ; still not precise :

The Goal

We need to query mathematical digital libraries to retrieve definitions, axioms and proved statements.

- Query-by-name : totally unreliable and not useful ; very quick
- Query-by-keyword+dc : it used to be the only practical way those old times when documents where unstructured ; not precise, but very quick
- Query-by-matching : it works for formal mathematics and for structured mathematics (content level) ; super-linear in the size of the library ; still not precise :

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

The Goal

We need to query mathematical digital libraries to retrieve definitions, axioms and proved statements.

- Query-by-name : totally unreliable and not useful ; very quick
- Query-by-keyword+dc : it used to be the only practical way those old times when documents where unstructured ; not precise, but very quick
- Query-by-matching : it works for formal mathematics and for structured mathematics (content level) ; super-linear in the size of the library ; still not precise :

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

vs

$$\forall m, n : nat. m + 1 > n \rightarrow (S\ m) > n$$

The Goal

We need to query mathematical digital libraries to retrieve definitions, axioms and proved statements.

- Query-by-name : totally unreliable and not useful ; very quick
- Query-by-keyword+dc : it used to be the only practical way those old times when documents where unstructured ; not precise, but very quick
- Query-by-matching : it works for formal mathematics and for structured mathematics (content level) ; super-linear in the size of the library ; still not precise :

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

vs

$$\forall m, n : nat.m + 1 > n \rightarrow (S\ m) > n$$

- Query-up-to-* : requires a decidable equivalence relation ; super-linear in the size of the library ; it may interact badly with the underlying conversion rules (e.g. extensional equality vs intensional equality)

The Problem

- In practice we need to combine all of the above methods.

The Problem

- In practice we need to combine all of the above methods.
- Some of them are super-linear in the size of the library.
- The libraries are supposed to be huge (e.g. Coq library is about 40.000 objects and it holds only a very few mathematical notions).
- Huge libraries are supposed to be distributed.

The Problem

- In practice we need to combine all of the above methods.
- Some of them are super-linear in the size of the library.
- The libraries are supposed to be huge (e.g. Coq library is about 40.000 objects and it holds only a very few mathematical notions).
- Huge libraries are supposed to be distributed.
- WE CAN NOT ITERATE OVER THE LIBRARY

The Problem

- In practice we need to combine all of the above methods.
- Some of them are super-linear in the size of the library.
- The libraries are supposed to be huge (e.g. Coq library is about 40.000 objects and it holds only a very few mathematical notions).
- Huge libraries are supposed to be distributed.
- WE CAN NOT ITERATE OVER THE LIBRARY
- One solution in the literature : Term Indexing Techniques
  - store all the terms in a data-structure that maximizes sharing
  - naturally exploit the sharing to speed-up unification and matching

The Solution

We can adopt a two-phases approach : filtering + matching

- data-mining on data : [BATCH] We extract from every matchable type a set of metadata which are related to the kind of match we are interested in.

The Solution

We can adopt a two-phases approach : filtering + matching

- data-mining on data : [BATCH] We extract from every matchable type a set of metadata which are related to the kind of match we are interested in.
- data-mining on the pattern : We extract from the pattern an incomplete set of constraints on the computed metadata.

The Solution

We can adopt a two-phases approach : filtering + matching

- data-mining on data : [BATCH] We extract from every matchable type a set of metadata which are related to the kind of match we are interested in.
- data-mining on the pattern : We extract from the pattern an incomplete set of constraints on the computed metadata.
- filtering : We compute the set of objects in the library whose metadata satisfy the above constraints.

The Solution

We can adopt a two-phases approach : filtering + matching

- data-mining on data : [BATCH] We extract from every matchable type a set of metadata which are related to the kind of match we are interested in.
- data-mining on the pattern : We extract from the pattern an incomplete set of constraints on the computed metadata.
- filtering : We compute the set of objects in the library whose metadata satisfy the above constraints.
- matching : We iterate the matching operation only on the computed set of candidates.

The Solution

We can adopt a two-phases approach : filtering + matching

- data-mining on data : [BATCH] We extract from every matchable type a set of metadata which are related to the kind of match we are interested in.
- data-mining on the pattern : We extract from the pattern an incomplete set of constraints on the computed metadata.
- filtering : We compute the set of objects in the library whose metadata satisfy the above constraints.
- matching : We iterate the matching operation only on the computed set of candidates.

If the filtering operation is both quick and correct and the set of candidates is small we achieve both accuracy and performance.

Use Case 1 : Lemma That Can Be Applied (1/5)

We want to know which theorems can be applied to prove a given goal.

Use Case 1 : Lemma That Can Be Applied (1/5)

We want to know which theorems can be applied to prove a given goal.

- We restrict ourselves to the case of first order matching (i.e. just one equation has metavariables in it and all the terms are rigid).

  Example : $(?a*?n \leq ?a*?m)$ matches $(2x \leq 2(y+1))$

  Example : $(?P\ ?n)$ does not match $\sqrt{5}$

Use Case 1 : Lemma That Can Be Applied (1/5)

We want to know which theorems can be applied to prove a given goal.

- We restrict ourselves to the case of first order matching (i.e. just one equation has metavariables in it and all the terms are rigid).
  Example : $(?a*?n \leq ?a*?m)$ matches $(2x \leq 2(y+1))$
  Example : $(?P\ ?n)$ does not match $\sqrt{5}$
- We identify the following set of metadata (both on the data and on the patterns) :
  - The constant in head position in the conclusion (if any)

Use Case 1 : Lemma That Can Be Applied (1/5)

We want to know which theorems can be applied to prove a given goal.

- We restrict ourselves to the case of first order matching (i.e. just one equation has metavariables in it and all the terms are rigid).
  Example : $(?a*?n \leq ?a*?m)$ matches $(2x \leq 2(y+1))$
  Example : $(?P\ ?n)$ does not match $\sqrt{5}$
- We identify the following set of metadata (both on the data and on the patterns) :
  - The constant in head position in the conclusion (if any)
  - The list of constants in other positions in the conclusion

Use Case 1 : Lemma That Can Be Applied (1/5)

We want to know which theorems can be applied to prove a given goal.

- We restrict ourselves to the case of first order matching (i.e. just one equation has metavariables in it and all the terms are rigid).
  Example : $(?a{*}?n \leq ?a{*}?m)$ matches $(2x \leq 2(y+1))$
  Example : $(?P\ ?n)$ does not match $\sqrt{5}$
- We identify the following set of metadata (both on the data and on the patterns) :
  - The constant in head position in the conclusion (if any)
  - The list of constants in other positions in the conclusion

  $2 * x \leq 2 * (y+1)$

Use Case 1 : Lemma That Can Be Applied (1/5)

We want to know which theorems can be applied to prove a given goal.

- We restrict ourselves to the case of first order matching (i.e. just one equation has metavariables in it and all the terms are rigid).
  Example : $(?a{*}?n \leq ?a{*}?m)$ matches $(2x \leq 2(y+1))$
  Example : $(?P\ ?n)$ does not match $\sqrt{5}$
- We identify the following set of metadata (both on the data and on the patterns) :
  - The constant in head position in the conclusion (if any)
  - The list of constants in other positions in the conclusion

  $2 * x \leq 2 * (y + 1)$

Use Case 1 : Lemma That Can Be Applied (1/5)

We want to know which theorems can be applied to prove a given goal.

- We restrict ourselves to the case of first order matching (i.e. just one equation has metavariables in it and all the terms are rigid).
  Example : $(?a*?n \leq ?a*?m)$ matches $(2x \leq 2(y+1))$
  Example : $(?P\ ?n)$ does not match $\sqrt{5}$
- We identify the following set of metadata (both on the data and on the patterns) :
  - The constant in head position in the conclusion (if any)
  - The list of constants in other positions in the conclusion

  $2 * x \leq 2 * (y+1)$ $\quad\quad$ $?a*?n \leq ?a*?m$

Use Case 1 : Lemma That Can Be Applied (1/5)

We want to know which theorems can be applied to prove a given goal.

- We restrict ourselves to the case of first order matching (i.e. just one equation has metavariables in it and all the terms are rigid).

  Example : $(?a {*} ?n \leq ?a {*} ?m)$ matches $(2x \leq 2(y+1))$

  Example : $(?P\ ?n)$ does not match $\sqrt{5}$
- We identify the following set of metadata (both on the data and on the patterns) :
  - The constant in head position in the conclusion (if any)
  - The list of constants in other positions in the conclusion

  $2 * x \leq 2 * (y+1)$ $\quad$ $?a {*} ?n \leq ?a {*} ?m$ $\quad$ $?a^2 {*} ?n \leq \sqrt{?b} {*} ?m$

Use Case 1 : Lemma That Can Be Applied (1/5)

We want to know which theorems can be applied to prove a given goal.

- We restrict ourselves to the case of first order matching (i.e. just one equation has metavariables in it and all the terms are rigid).
  Example : $(?a{*}?n \leq ?a{*}?m)$ matches $(2x \leq 2(y+1))$
  Example : $(?P\ ?n)$ does not match $\sqrt{5}$
- We identify the following set of metadata (both on the data and on the patterns) :
  - The constant in head position in the conclusion (if any)
  - The list of constants in other positions in the conclusion

$2 * x \leq 2 * (y+1)$ $\qquad$ $?a{*}?n{\leq}?a{*}?m$ $\qquad$ $?a^2{*}?n{\leq}\sqrt{?b}{*}?m$

$\leq * + 2\ 1$ $\qquad$ $\leq *$ $\qquad$ $\leq * \sqrt{\_}\ \_^2$

Use Case 1 : Lemma That Can Be Applied (2/5)

- The query “give me all the theorems whose metadata are a subset of the constraints” is complete but not effective (to check the condition we have to iterate over the whole library)

Use Case 1 : Lemma That Can Be Applied (2/5)

- The query “give me all the theorems whose metadata are a subset of the constraints” is complete but not effective (to check the condition we have to iterate over the whole library)
- We trade completeness for efficiency introducing two sets of constraints :
  - “Only” Constraints : they are the constraints seen before. We look for theorems whose metadata are subsets of the “only” constraints.

Use Case 1 : Lemma That Can Be Applied (2/5)

- The query "give me all the theorems whose metadata are a subset of the constraints" is complete but not effective (to check the condition we have to iterate over the whole library)
- We trade completeness for efficiency introducing two sets of constraints :
  - "Only" Constraints : they are the constraints seen before. We look for theorems whose metadata are subsets of the "only" constraints.
  - "Must" Constraints : they are a subset of the "only" constraints. We look for theorems whose metadata are a superset of the "must" constraints. Their computation is very efficient.

Use Case 1 : Lemma That Can Be Applied (3/5)

- Example : $2 * x \leq 2 * (y + 1)$

  Must : $\leq$ $*$ Only : $\leq$ $*$ $+$ 2 1

  Good match (OK) : $?a{*}?n{\leq}?a{*}?m$

Use Case 1 : Lemma That Can Be Applied (3/5)

- Example : $2 * x \leq 2 * (y + 1)$

  Must : $\leq$ $*$ Only : $\leq$ $*$ $+$ 2 1

  Good match (OK) : $?a{*}?n{\leq}?a{*}?m$

  No match (OK) : $?a^2{*}?n{\leq}\sqrt{?b}{*}?m$

Use Case 1 : Lemma That Can Be Applied (3/5)

- Example : $2 * x \leq 2 * (y + 1)$

  Must : $\leq *$ Only : $\leq * + 2\ 1$

  Good match (OK) : $?a*?n \leq ?a*?m$

  No match (OK) : $?a^2*?n \leq \sqrt{?b}*?m$

  False match : $1*?n \leq ?n$

Use Case 1 : Lemma That Can Be Applied (3/5)

- Example : $2 * x \leq 2 * (y + 1)$

  Must : $\leq *$ Only : $\leq * + 2\ 1$

  Good match (OK) : $?a{*}?n{\leq}?a{*}?m$

  No match (OK) : $?a^2{*}?n{\leq}\sqrt{?b}{*}?m$

  False match : $1{*}?n{\leq}?n$

  No match (ERROR) : $?a\ ?o\ ?n \leq ?a\ ?o\ (?b + 1)$

Use Case 1 : Lemma That Can Be Applied (4/5)

The (simplified) query generated for :

Must : $\leq$ $*$ Only : $\leq$ $*$ $+$ 2 1

```
let S =
  select every t in  the library such that
    t.head = '≤' and '∗' occurs in t.in_conclusion
in
  select every t in S such that
    t.in_conclusion subset of {≤,∗, +, 2, 1}
```

Use Case 1 : Lemma That Can Be Applied (4/5)

The (simplified) query generated for :

Must : $\leq *$ Only : $\leq * + 2\ 1$

```
let S =
  select every t in  the library such that
    t.head = '≤' and '∗' occurs in t.in_conclusion
in
  select every t in S such that
    t.in_conclusion subset of {≤,∗, +, 2, 1}
```

The first select requires a query to the DB for each constraint in the must list.
(Cheap)

Use Case 1 : Lemma That Can Be Applied (4/5)

The (simplified) query generated for :

Must : $\leq *$ Only : $\leq * + 2\ 1$

```
let S =
  select every t in the library such that
    t.head = '≤' and '∗' occurs in t.in_conclusion
in
  select every t in S such that
    t.in_conclusion subset of {≤,∗, +, 2, 1}
```

The first select requires a query to the DB for each constraint in the must list. (Cheap)

The second select requires a query to the DB for each object in the result of the first query. (Expensive if the “must” constraints are not tight.)

Use Case 1 : Lemma That Can Be Applied (5/5)

Demo

Use Case 2 : Elimination Principles (1/4)

We want to know which are the elimination principles available for a given datatype/proposition inductively defined.

Use Case 2 : Elimination Principles (1/4)

We want to know which are the elimination principles available for a given datatype/proposition inductively defined.

We restrict ourselves to those elimination principles on T whose shape is :

$$\forall \vec{x} : \vec{S}.\ \forall P : T \rightarrow Sort.\ \forall \vec{y} : \vec{S'}.\ \forall x : T. \forall \vec{z} : \vec{S''}.\ (P\ x)$$

Use Case 2 : Elimination Principles (1/4)

We want to know which are the elimination principles available for a given datatype/proposition inductively defined.

We restrict ourselves to those elimination principles on T whose shape is :

$$\forall \vec{x} : \vec{S}.\ \forall P : T \to Sort.\ \forall \vec{y} : \vec{S'}.\ \forall x : T. \forall \vec{z} : \vec{S''}.\ (P\ x)$$

Example (induction principle) :

$$\begin{array}{l} \forall P : nat \to Prop. \\ (P\ O) \to (\forall n : nat.(P\ n) \to (P\ (S\ n))) \to \\ \forall n : nat.(P\ n) \end{array}$$

Use Case 2 : Elimination Principles (1/4)

We want to know which are the elimination principles available for a given datatype/proposition inductively defined.

We restrict ourselves to those elimination principles on T whose shape is :

$$\forall \vec{x} : \vec{S}.\ \forall P : T \rightarrow Sort.\ \forall \vec{y} : \vec{S'}.\ \forall x : T. \forall \vec{z} : \vec{S''}.\ (P\ x)$$

Example (induction principle) :

$$\begin{array}{l} \forall P : nat \rightarrow Prop. \\ (P\ O) \rightarrow (\forall n : nat.(P\ n) \rightarrow (P\ (S\ n))) \rightarrow \\ \forall n : nat.(P\ n) \end{array}$$

Example (even-odd complementarity) :

$$\begin{array}{l} \forall P : nat \rightarrow Prop. \\ (\forall n : nat.(Even\ n) \rightarrow (P\ n)) \rightarrow (\forall n : nat.(Odd\ n) \rightarrow (P\ n)) \rightarrow \\ \forall n : nat.(P\ n) \end{array}$$

Use Case 2 : Elimination Principles (2/4)

- We extract from the pattern the following set of constraints :
  - The exact sort of the property we are interested in.

Use Case 2 : Elimination Principles (2/4)

- We extract from the pattern the following set of constraints :
  - The exact sort of the property we are interested in.
  - The fact that it must occur in the head position of an hypothesis which is a product of length 1.

Use Case 2 : Elimination Principles (2/4)

- We extract from the pattern the following set of constraints :
  - The exact sort of the property we are interested in.
  - The fact that it must occur in the head position of an hypothesis which is a product of length 1.
  - The fact that the type must occur in the head position of an hypothesis which is a product of length 0 and and also in another hypothesis (not in main position).

Use Case 2 : Elimination Principles (2/4)

- We extract from the pattern the following set of constraints :
  - The exact sort of the property we are interested in.
  - The fact that it must occur in the head position of an hypothesis which is a product of length 1.
  - The fact that the type must occur in the head position of an hypothesis which is a product of length 0 and and also in another hypothesis (not in main position).
  - The fact that the head of the conclusion must be an occurrence of a bound variable (a Rel).

Use Case 2 : Elimination Principles (2/4)

- We extract from the pattern the following set of constraints :
  - The exact sort of the property we are interested in.
  - The fact that it must occur in the head position of an hypothesis which is a product of length 1.
  - The fact that the type must occur in the head position of an hypothesis which is a product of length 0 and and also in another hypothesis (not in main position).
  - The fact that the head of the conclusion must be an occurrence of a bound variable (a Rel).

$$\forall P : nat \rightarrow Prop. \forall n : nat. \ (P \ n)$$

Use Case 2 : Elimination Principles (2/4)

- We extract from the pattern the following set of constraints :
  - The exact sort of the property we are interested in.
  - The fact that it must occur in the head position of an hypothesis which is a product of length 1.
  - The fact that the type must occur in the head position of an hypothesis which is a product of length 0 and and also in another hypothesis (not in main position).
  - The fact that the head of the conclusion must be an occurrence of a bound variable (a Rel).

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

Use Case 2 : Elimination Principles (2/4)

- We extract from the pattern the following set of constraints :
  - The exact sort of the property we are interested in.
  - The fact that it must occur in the head position of an hypothesis which is a product of length 1.
  - The fact that the type must occur in the head position of an hypothesis which is a product of length 0 and and also in another hypothesis (not in main position).
  - The fact that the head of the conclusion must be an occurrence of a bound variable (a Rel).

$$\forall P : nat \rightarrow Prop.\ \forall n : nat.\ (P\ n)$$

nat, Prop(1), nat(0), Rel(_)

Use Case 2 : Elimination Principles (2/4)

- We extract from the pattern the following set of constraints :
  - The exact sort of the property we are interested in.
  - The fact that it must occur in the head position of an hypothesis which is a product of length 1.
  - The fact that the type must occur in the head position of an hypothesis which is a product of length 0 and and also in another hypothesis (not in main position).
  - The fact that the head of the conclusion must be an occurrence of a bound variable (a Rel).

$$\forall P : nat \rightarrow Prop. \forall n : nat. (P\ n)$$

nat, Prop(1), nat(0), Rel(_)   “must” constraints

Since any other constant can appear in the statement (e.g. Even/Odd), we do not impose any “only” constraints.

Use Case 2 : Elimination Principles (3/4)

The (very simplified) query generated for :

Must : nat, Prop(1), nat(0), Rel(_)

Use Case 2 : Elimination Principles (3/4)

The (very simplified) query generated for :

Must : nat, Prop(1), nat(0), Rel(_)

select every t in the library such that

Prop occurs in head position at depth 1

in an hypothesis of t and

'nat' occurs in head position at depth 0

in an hypothesis of t and

'nat' occurs not in head position

in an hypothesis of t and

a bound variable occurs in head position

in the conclusion of t

Use Case 2 : Elimination Principles (3/4)

The (very simplified) query generated for :

Must : nat, Prop(1), nat(0), Rel(_)

select every t in the library such that
Prop occurs in head position at depth 1
in an hypothesis of t and
'*nat*' occurs in head position at depth 0
in an hypothesis of t and
'*nat*' occurs not in head position
in an hypothesis of t and
a bound variable occurs in head position
in the conclusion of t

The number of queries to the underling DB is fixed (thus the query is cheap).

Use Case 2 : Elimination Principles (4/4)

Demo

The General Case (1/3)

We start from a pattern and we identify the following set of constraints :

- For each occurrence of a constant, its position $\in$ {MainConclusion, InConclusion, MainHypothesis, InHypothesis} and its URI

The General Case (1/3)

We start from a pattern and we identify the following set of constraints :

- For each occurrence of a constant, its position $\in$ {MainConclusion, InConclusion, MainHypothesis, InHypothesis} and its URI
- For each occurrence of a bound variable, its position $\in$ {MainConclusion, MainHypothesis}

The General Case (1/3)

We start from a pattern and we identify the following set of constraints :

- For each occurrence of a constant, its position $\in$ {MainConclusion, InConclusion, MainHypothesis, InHypothesis} and its URI
- For each occurrence of a bound variable, its position $\in$ {MainConclusion, MainHypothesis}
- For each occurrence of a sort, its position $\in$ {MainConclusion, MainHypothesis} and its type $\in$ {Prop, Set, Type}

The General Case (1/3)

We start from a pattern and we identify the following set of constraints :

- For each occurrence of a constant, its position $\in$ {MainConclusion, InConclusion, MainHypothesis, InHypothesis} and its URI
- For each occurrence of a bound variable, its position $\in$ {MainConclusion, MainHypothesis}
- For each occurrence of a sort, its position $\in$ {MainConclusion, MainHypothesis} and its type $\in$ {Prop, Set, Type}
- For each occurrence $\in$ {MainConclusion, MainHypothesis} we also record its depth, i.e. the number of products in the type/hypothesis

$\forall S : Set.\forall P : S \rightarrow Prop.(\forall x : S.\forall y : S.(P\ x\ y) \rightarrow (P\ y\ x)) \rightarrow (reflexive\ S\ P)$

The General Case (1/3)

We start from a pattern and we identify the following set of constraints :

- For each occurrence of a constant, its position $\in$ {MainConclusion, InConclusion, MainHypothesis, InHypothesis} and its URI
- For each occurrence of a bound variable, its position $\in$ {MainConclusion, MainHypothesis}
- For each occurrence of a sort, its position $\in$ {MainConclusion, MainHypothesis} and its type $\in$ {Prop, Set, Type}
- For each occurrence $\in$ {MainConclusion, MainHypothesis} we also record its depth, i.e. the number of products in the type/hypothesis

$\forall S : Set.\forall P : S \to Prop.(\forall x : S.\forall y : S.(P\ x\ y) \to (P\ y\ x)) \to (reflexive\ S\ P)$

The General Case (1/3)

We start from a pattern and we identify the following set of constraints :

- For each occurrence of a constant, its position $\in$ {MainConclusion, InConclusion, MainHypothesis, InHypothesis} and its URI
- For each occurrence of a bound variable, its position $\in$ {MainConclusion, MainHypothesis}
- For each occurrence of a sort, its position $\in$ {MainConclusion, MainHypothesis} and its type $\in$ {Prop, Set, Type}
- For each occurrence $\in$ {MainConclusion, MainHypothesis} we also record its depth, i.e. the number of products in the type/hypothesis

$\forall S : Set.\forall P : S \to Prop.(\forall x : S.\forall y : S.(P\ x\ y) \to (P\ y\ x)) \to (reflexive\ S\ P)$

Set(0), Prop(1), Rel(3), reflexive(3)

The General Case (1/3)

We start from a pattern and we identify the following set of constraints :

- For each occurrence of a constant, its position $\in$ {MainConclusion, InConclusion, MainHypothesis, InHypothesis} and its URI
- For each occurrence of a bound variable, its position $\in$ {MainConclusion, MainHypothesis}
- For each occurrence of a sort, its position $\in$ {MainConclusion, MainHypothesis} and its type $\in$ {Prop, Set, Type}
- For each occurrence $\in$ {MainConclusion, MainHypothesis} we also record its depth, i.e. the number of products in the type/hypothesis

$\forall S : Set.\forall P : S \to Prop.(\forall x : S.\forall y : S.(P\ x\ y) \to (P\ y\ x)) \to (reflexive\ S\ P)$

Set(0), Prop(1), Rel(3), reflexive(3)

The General Case (2/3)

- A. The wanted objects must have a reference to a given object R (or to a given primitive constant S or to a bound variable) in a given position P with a given depth index D.

- B. The wanted objects may have a reference to an object (or to a primitive constant or to a bound variable) only if its position is not included in a given set U of positions, or if it concerns a given object R (or a primitive constant S, or a bound variable) in a given position P with a given depth index D.

The parameters R, S, P, D, U are optional.

The General Case (3/3)

We always generate the query in a uniform way. Different results are obtained imposing different subsets of the constraints :

- Use Case 1 : Searching for a lemma. We impose “must” and “only” constraints only on constants and only on the conclusion.

The General Case (3/3)

We always generate the query in a uniform way. Different results are obtained imposing different subsets of the constraints :

- Use Case 1 : Searching for a lemma. We impose "must" and "only" constraints only on constants and only on the conclusion.
- Use Case 2 : Searching for an induction principle. We impose only the "must" constraints. We relax the constraint on the depth of the Rel in MainConclusion to any depth.

The General Case (3/3)

We always generate the query in a uniform way. Different results are obtained imposing different subsets of the constraints :

- Use Case 1 : Searching for a lemma. We impose "must" and "only" constraints only on constants and only on the conclusion.
- Use Case 2 : Searching for an induction principle. We impose only the "must" constraints. We relax the constraint on the depth of the Rel in MainConclusion to any depth.
- Use Case 3 : Searching for relations or functions. Demo

The General Case (3/3)

We always generate the query in a uniform way. Different results are obtained imposing different subsets of the constraints :

- Use Case 1 : Searching for a lemma. We impose "must" and "only" constraints only on constants and only on the conclusion.
- Use Case 2 : Searching for an induction principle. We impose only the "must" constraints. We relax the constraint on the depth of the Rel in MainConclusion to any depth.
- Use Case 3 : Searching for relations or functions. Demo
- Use Case 4 : General concepts. Demo

The General Case (3/3)

We always generate the query in a uniform way. Different results are obtained imposing different subsets of the constraints :

- Use Case 1 : Searching for a lemma. We impose "must" and "only" constraints only on constants and only on the conclusion.
- Use Case 2 : Searching for an induction principle. We impose only the "must" constraints. We relax the constraint on the depth of the Rel in MainConclusion to any depth.
- Use Case 3 : Searching for relations or functions. Demo
- Use Case 4 : General concepts. Demo
- Use Case 5 : Other queries ?

The General Case (3/3)

We always generate the query in a uniform way. Different results are obtained imposing different subsets of the constraints :

- Use Case 1 : Searching for a lemma. We impose “must” and “only” constraints only on constants and only on the conclusion.
- Use Case 2 : Searching for an induction principle. We impose only the “must” constraints. We relax the constraint on the depth of the Rel in MainConclusion to any depth.
- Use Case 3 : Searching for relations or functions. Demo
- Use Case 4 : General concepts. Demo
- Use Case 5 : Other queries ?

There are several other queries that are not instantiation of this general pattern, but that can be expressed in the underlying query language.

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach
- Term Indexing Techniques :
  - More sharing $\Rightarrow$ less storage space required

- Approach based on Metadata :

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach
- Term Indexing Techniques :
  - More sharing $\Rightarrow$ less storage space required
  - Unification : speed-up of one order of magnitude

- Approach based on Metadata :

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach
- Term Indexing Techniques :
  - More sharing $\Rightarrow$ less storage space required
  - Unification : speed-up of one order of magnitude
  - Complex data-structures $\Rightarrow$ require ad-hoc implementations
- Approach based on Metadata :

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach
- Term Indexing Techniques :
  - More sharing $\Rightarrow$ less storage space required
  - Unification : speed-up of one order of magnitude
  - Complex data-structures $\Rightarrow$ require ad-hoc implementations
- Approach based on Metadata :

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach
- Term Indexing Techniques :
  - More sharing $\Rightarrow$ less storage space required
  - Unification : speed-up of one order of magnitude
  - Complex data-structures $\Rightarrow$ require ad-hoc implementations
- Approach based on Metadata :
  - A very light approximation in a standard format (RDF triples) $\Rightarrow$ we can adopt standard DB technology (relational DB)

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach
- Term Indexing Techniques :
  - More sharing $\Rightarrow$ less storage space required
  - Unification : speed-up of one order of magnitude
  - Complex data-structures $\Rightarrow$ require ad-hoc implementations
- Approach based on Metadata :
  - A very light approximation in a standard format (RDF triples) $\Rightarrow$ we can adopt standard DB technology (relational DB)
  - Transparent distribution achievable via distributed DBs

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach
- Term Indexing Techniques :
  - More sharing $\Rightarrow$ less storage space required
  - Unification : speed-up of one order of magnitude
  - Complex data-structures $\Rightarrow$ require ad-hoc implementations
- Approach based on Metadata :
  - A very light approximation in a standard format (RDF triples) $\Rightarrow$ we can adopt standard DB technology (relational DB)
  - Transparent distribution achievable via distributed DBs
  - Does not support unification only : an open model to independently add new metadata, spiders and clients (e.g. query up-to-isomorphisms)

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach
- Term Indexing Techniques :
  - More sharing $\Rightarrow$ less storage space required
  - Unification : speed-up of one order of magnitude
  - Complex data-structures $\Rightarrow$ require ad-hoc implementations
- Approach based on Metadata :
  - A very light approximation in a standard format (RDF triples) $\Rightarrow$ we can adopt standard DB technology (relational DB)
  - Transparent distribution achievable via distributed DBs
  - Does not support unification only : an open model to independently add new metadata, spiders and clients (e.g. query up-to-isomorphisms)
  - Unification is often too strict ; false matches may be interesting

Comparison With Term Indexing Techniques

- e.g. discrimination trees, substitution trees, coded context trees
- Not incompatible : they are the limit case of our approach
- Term Indexing Techniques :
  - More sharing $\Rightarrow$ less storage space required
  - Unification : speed-up of one order of magnitude
  - Complex data-structures $\Rightarrow$ require ad-hoc implementations
- Approach based on Metadata :
  - A very light approximation in a standard format (RDF triples) $\Rightarrow$ we can adopt standard DB technology (relational DB)
  - Transparent distribution achievable via distributed DBs
  - Does not support unification only : an open model to independently add new metadata, spiders and clients (e.g. query up-to-isomorphisms)
  - Unification is often too strict ; false matches may be interesting