An Indexer and Query Language for Libraries written in LambdaPi/Dedukti

Claudio Sacerdoti Coen

<claudio.sacerdoticoen@unibo.it>

University of Bologna

13/09/2024

Outline

Specific challenges for Dedukti/Lambdapi indexing and retrieval

Contribution to LambdaPi code

Implementation

Dedukti/LambdaPi (LF modulo)

- types are identified up to the symmetric-transitive closure of rewriting rules

  example: $\vdash I : True$ and $2 < 3 \hookrightarrow True$;
  therefore $\vdash I : 2 < 3$

- greatly simplifies LF encodings

  example: $El\ (arrow\ A\ A) \hookrightarrow El\ A\ \rightarrow\ El\ A$
  therefore $\vdash \lambda x : A.x\ :\ El\ (arrow\ A\ A)$

- makes indexing, retrieval and alignment between libraries much harder

  Indexes/search should be up to as well
  example (Coqine's output): $El\ (arrow\ \mathbb{N}\ \mathbb{N}) \rightarrow \mathbb{N}$

Libraries in Dedukti/LambdaPi

When exporting the library of an ITP to Dedukti, you get the encodings of the statements.

example:

```
def fact :
  __ : cic.Term univs.Typez nat ->
    cic.Term univs.Typez nat
```

but a user is likely to look for just `nat -> nat`

Indexing/search should be up to encoding!
problem: encodings are user defined

Libraries and alignments

In Dedukti you can have libraries coming from multiple systems/theories/encodings.

Open problem: how to combine results from two libraries?

Preliminary problem: how to look for statements in multiple libraries where concepts have different definitions and shapes?

Indexing/search should be up to alignments!
Example: looking for $r = n/m$ one should retrieve Coq's $r = div\ n\ m\ (p : m \neq 0)$, Isabelle's $r = div\ n\ m$ and Abella's $div\ n\ m\ r$

Alignments can be complex and they are also user defined

Dedukti to the rescue!

Rewriting is also the solution:

- Indexing/searching up to rewriting rules is approximated indexing normal forms of terms
  example: index *El* (*arrow* $\mathbb{N}\ \mathbb{N}) \rightarrow \mathbb{N}$ as $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$
- Indexing/searching up to encodings is implemented by using user provided rewrite rules to "undo" the encoding
  example: "*El* $\$T \hookrightarrow \$T$" or "*cic.Term* _ $\$T \hookrightarrow \$T$"
- Indexing/searching up to alignment is implemented by using user provided rewrite rules to map terms to "Wikipedia"/content forms
  example:
  "*Coq.ConstructiveReals.div* $\$n\ \$m$ _ $\hookrightarrow$ *Wikipedia.div* $\$n\ \$m$"

Wait a minute!

Consider again the rule to undo the rewriting:

$$El\ \$T \hookrightarrow \$T$$

The right hand side is not well typed!
The rule is rejected

We cannot reuse the rewriting machinery of Dedukti/LambdaPi, unless we relax all checks.

Outline

Specific challenges for Dedukti/Lambdapi indexing and retrieval

Contribution to LambdaPi code

Implementation

What has been implemented

In LambdaPi 2.4.0 (2023-07-28):

- `lambdapi index --rules filerules [--add] file1 ... filen`
  - creates an index for the given Dedukti/LambdaPi files
  - `filerules` contains the (untyped, unqualified) rewriting rules for indexing up-to
- `lambdapi search query`
  - runs a query against the index
  - there is also an interactive LambdaPi command to query the library
- `lambdapi websearch [--port=PORT]`
  - runs a local search engine that can be interrogated using a standard browser (default port 808)

Scenarios for Formula Retrieval

Why should we be interested in searching for a formula?

- Scenario 1: searching the current development
  - We have some theories already loaded and we are developing a new one;
  - we want to retrieve theorems from them;
  - the foundation is typically just one;
  - typical motivation: to apply the result in a proof step (e.g. during automation);
    example: find a theorem whose conclusion generalizes
    $3x + y > f(x) + g(y)$
    possible result:
    $\forall x, y, x', y'. x > x' \wedge y \geq y' \Rightarrow x + y > x' + y'$
  - interested in speed and therefore high precision if done automatically;
  - the query must match precisely and it is usually espressed as a pattern up to instantiation/generalization

Scenarios for Formula Retrieval

Why should we be interested in searching for a formula?

- Scenario 2: searching in every library
  - We want to look for a theorem inside all libraries;
  - heterogeneous foundations;
  - the libraries have not been loaded;
  - interested in high recall;
  - low precision is a feature: we want to find related theorems;
  - example: find a statement that speaks about groups, has as a premise *is_normal* _ and whose conclusion contains $|\_|$;
  - queries are written in a query language; atomic queries are patterns up to instantiation/generalization or restrictions based on metadata

Scenarios for Formula Retrieval

We focused on scenario 2: searching in every library only

- every user has just one global persistent index that is manually updated
- the current LambdaPi devel. is not indexed/kept in sync
- query language:

```
Q ::= B | Q,Q | Q;Q | Q|PATH
B ::= WHERE HOW GENERALIZE? PATTERN
PATH ::= << string >>
WHERE ::= name
        | anywhere
        | rule | lhs | rhs
        | type | concl | hyp | spine
HOW ::= > | = | >=
GENERALIZE ::= generalize
PATTERN ::= << term possibly containing
              placeholders _ (for terms)
              and V# (for variables) >>
```

The Query Language by Examples

Examples:

- `name = nat`
  all constants whose name is `nat`
- `name = nat | matita_arithmetics`
  all constants whose name is `nat` that are defined in the arithmetical library of Matita
- `concl = nat`
  all constants whose type is of the form $\overrightarrow{\Pi x_i : T_i}.C$
  and C is not a product
  and C matches the pattern `nat`
  example:

```
def gcd :
  __ : cic.Term univs.Typez matita_arithmetics_nat.nat ->
  __1 : cic.Term univs.Typez matita_arithmetics_nat.nat ->
  cic.Term univs.Typez matita_arithmetics_nat.nat
```

`nat` in the pattern has been disambiguated first to `matita_arithmetics_nat.nat` via the implicit query `name = nat`

The Query Language by Examples

Examples:

- `concl = plus O _`
  the conclusion must match exactly the pattern `plus O _`
  NO MATCH!
- `concl >= plus O _`
  a subterm of the conclusion must match the pattern `plus O _`
  example: it matches the theorem $\forall n. plus\ O\ n = n$
- `concl = generalize (plus 3 4 = plus 4 3)`
  the conclusion must be a generalization of the pattern `plus 3 4 = plus 4 3`
  example: it matches the theorem
  $\forall n, m. plus\ n\ m = plus\ m\ n$ because n and m are universally quantified

The Query Language

- Basic queries can be combined with conjunctions/disjunctions (Prolog syntax)
- Many kind of positions to identify parts of a type/rule
  example: in $\overrightarrow{\Pi x_i : T_i}.C$
  - the whole formula is a type
  - each T_i is an hypothesis and
  - $\overrightarrow{\Pi x_{i+n} : T_{i+n}}.C$ is a spine
- Rewriting rules are also indexed with positioning information (lhs, rhs, anywhere)

Outline

Specific challenges for Dedukti/Lambdapi indexing and retrieval

Contribution to LambdaPi code

Implementation

Indexing

An index is made of:

- a map from constant names to sets of identifiers
  example: `nat` ↦ `matita_arithmetics_nat.nat`
  used also to disambiguate constants in queries
- a discrimination tree mapping terms to sets of (identifiers × positions)
  example: `plus O #V` ↦
  `<matita_arithmetics_nat.plus_O_n, conclusion>`

  - terms are normalized according to the user rules before computing the subterms
  - every subterm of the types of constants/sides of rules is indexed
  - all variables are mapped to the same placeholder `#V` during indexing
  - all subterms are indexed twice: once normally, once replacing variables universally quantified in the spine with a "don't care" placeholder _
    used by generalization basic queries

Conclusions

- Tested on the standard libraries of Matita and HOL-Light
- Rewriting rules to undo those two encodings implemented (6 lines in total)
- Size of the combined index on disk: 20MB (6MB Matita + 14MB HOL)
- Indexing time is reasonably fast (few minutes)

- Future work:
  - experiment with alignments
  - experiment with other systems/libraries
  - integrate in proof search