Integration of general-purpose automated theorem provers in Lean

Gabriel Ebner
Formal Methods in Mathematics
2020-01-08

Vrije Universiteit Amsterdam
Introduction
Premise selection
Applicative translation to FOL
Monomorphizing translation to HOL
Empirical results
Demonstration
Conclusion
Hammers

• “Magic button that proves all your theorems”

• e.g. Sledgehammer for Isabelle/HOL
  • popular, also: HOLyHammer, CoqHammer, etc.

• User-friendly integration of automated reasoning tools in proof assistants
example \((x \ y \ z : \text{nat}) : x.\gcd \ y \mid (x^*z).\gcd \ y \ := \)

by hammer

General purpose: should work for anything, no setup
1. Find already proven lemmas that look “useful” ("premise selection", "relevance filter")

2. Pass lemmas and goal to efficient external prover (e.g. Vampire, E, etc.)
   • Requires encoding into logic of prover

3. Import generated proof
   • Popular strategy: mine names of used lemmas, and reconstruct using slow prover
Introduction

Premise selection

Applicative translation to FOL

Monomorphizing translation to HOL

Empirical results

Demonstration

Conclusion
(Based on approach in CoqHammer (Czaja, Kaliszyk 2018))

Assign to every lemma a set of features based on its type:

• Every constants $c$ that occurs in the type
• The pair $(f, g)$ for every subterm $fa_1 \ldots (g \ldots ) \ldots a_n$

Ignore:

• eq, and, ...
• Type classes, and type class instance arguments.
• Cosine similarity with TF-IDF (term frequency-inverse document frequency)
  • Common way to calculate similarity between documents (= sequence/set of words) with lots of variations.
  • Here: document = lemma, word = feature.

1. Assign to every lemma the characteristic function of its feature set \( \in \mathbb{R}^{|F|} \)
2. Scale each coordinate by how rarely it occurs globally
3. Compute similarity of \( a \) and \( b \) as \( \frac{a \cdot b}{\|a\|\|b\|} \)

• Implemented in C++ (for performance reasons)
Issue: type classes

```
theorem le_of_lt { α} [preorder α] {a b : α} :
  a < b \rightarrow a \leq b :=
sorry

example (a b : nat) : ¬ a < b \lor a \leq b :=
by hammer

  • Should find le_of_lt because it talks about the preorder nat, even
  though the name preorder does not occur in the goal.

theorem le_of_lt' {a b : nat} : a < b \rightarrow a \leq b :=
sorry

  • Should not prefer le_of_lt' either.
```
Introduction
Premise selection

**Applicative translation to FOL**

Monomorphizing translation to HOL

Empirical results
Demonstration

Conclusion
Applicative translation

Translation to single-sorted first-order logic, like CoqHammer:

- Binary function $a(x, y)$ for application $xy$
- Predicate $p(x)$: (proposition) $x$ is inhabited
- Relation $t(x, y)$: $x$ has type $y$
- Equality is translated as equality.
- Constant $s$ means Type $u$.

For each constant to be exported, we write one formula expressing its type.
Example translation

```latex
\textbf{theorem} \texttt{nat.le\_succ} : \forall (n : \texttt{nat}),
\texttt{@\_has\_le\_le.\{0\} \texttt{nat \_nat\_has\_le} n (\texttt{nat\_succ} n)
\forall n, t(n, \texttt{nat}) \rightarrow p(a(a(a(\texttt{has\_le\_le}, \texttt{nat}), \texttt{nat\_has\_le}), n), a(\texttt{nat\_succ}, n))}
```
Example translation

```plaintext
theorem nat.le_succ : \forall (n : nat),
 \has_le.le.{0} nat nat.has_le n (nat.succ n)

\forall n,t(n,nat) \to p(a(a(a(has_le.le,nat),nat.has_le),n),a(nat.succ,n)))

fof(cnat_o_le__succ, axiom,
  (![Xn_n3]: (t(Xn_n3, cnat) \to p(a(a(a(chas__le_o_le, cnat ), cnat_o_has__le), Xn_n3), a(cnat_o_succ, Xn_n3))))).
```
Translation is unsound (= does not preserve unprovability).

→ “spurious” proofs

Two main reasons:

1. Definitional equality and propositional equality are identified.
2. Type $u$ and Type $(u+1)$ are identified.
We often need to show that two type class instances are equal.

E.g. if you want to apply \texttt{le_refl} to natural numbers:

\[
p(a(a(a(a(chas__le_o_le, X_{ga\_n2}),
\text{Xa\_n4}),
\text{Xa\_n4}))
\]

vs.

\[
p(a(a(a(a(chas__le_o_le, cnat),
\text{cnat\_o\_has\_le}),
\text{Xx\_n18}),
\text{Xx\_n18}))
\]

→ Heuristically add extra equations relating type class instances.
Introduction

Premise selection

Applicative translation to FOL

Monomorphizing translation to HOL

Empirical results

Demonstration

Conclusion
Simply-typed higher-order logic

Types:

- Booleans
- Base types: \(\text{nat}, \text{list nat}, \text{etc.}\)
- Function types: \(\tau_1 \rightarrow \tau_2\)

Terms (formulas are terms of Boolean type):

- Constants: \(\text{nat.add}, \text{etc.}\)
- Application: \(ts\)
- Variable: \(x\)
- Lambdas: \(\lambda x \, t\)

(We use closed Lean expressions as names for constants and base types.)
Two phases

Lean \(\xrightarrow{\text{abstraction}}\) HOL \(\xrightarrow{\text{type instantiation}}\) HOL

- sound translation
- enables provers to do non-first-order reasoning
  - built-in support for \(\mathbb{N}, \mathbb{Z}, \mathbb{R}, \ldots\)
  - synthesize lambdas
  - induction
- solves type class coherence issue
- mitigates issue with type classes in relevance filter
Abstraction

Turn $\forall \{\alpha : \text{Type } u\} [\text{preorder } \alpha] \ (a : \alpha), \ a \leq a$ into $\forall \ a : \ ?m_1', \ '@\text{has_le.le } \ ?m_1 \ ?m_2.\text{to_has_le'} \ a \ a$

- Replace non-HOL subterms by HOL constants.
  - dependent applications
  - pi types
  - types like list nat
  - ...

- Instance-implicit arguments are also included in the constants.
Type instantiation

Turn $\forall a : \text{"?m_1"}, \text{"@has_le.le ?m_1 ?m_2.to_has_le" } a a$
into $\forall a : \text{"nat"}, \text{"@has_le.le nat nat.preorder.to_has_le" } a a$

- Unify the constants in the HOL terms
  - $\text{"@has_le.le ?m_1 ?m_2.to_has_le"}$ occurs in lemma
  - $\text{"@has_le.le nat nat.has_le"}$ occurs in goal
  \[ \rightarrow \text{ Instantiate lemma by unifying } ?m_1 =?= \text{nat} \text{ and } ?m_2.to_has_le =?= \text{nat.has_le}. \]
- Also solves additional type-class constraints. E.g. a lemma about Archimedean fields might have an assumption archimedian $\alpha$ which does not occur in any constant.
Limitations

• equality between types: \( m = n \rightarrow \text{zmod } m = \text{zmod } n \)
  
  \( \rightarrow \) Bundle the non-type arguments? That is, translate to \( \Sigma n, \text{zmod } n \).

• dependent families: \( \forall i, \text{fin } i \) is translated to a base type

• proof arguments:
  \( @\text{roption.get} : \forall \alpha, \forall o : \text{roption } \alpha, o.\text{dom } \rightarrow \alpha \)
  
  • Just elide them? (Only affects nonemptiness of \( \alpha \) here.)
Introduction

Premise selection

Applicative translation to FOL

Monomorphizing translation to HOL

Empirical results

Demonstration

Conclusion
• Basic relevance filter (in C++)

• Applicative first-order encoding
  • interfaces with Vampire

• HOL encoding
  • interfaces with E-HO

• Proof reconstruction using super
  • Small superposition prover written in (meta-)Lean
• 31112 theorems in mathlib + core (everything that’s a declaration.thm)

• Try tactic at the same point in the file as the theorem.
  • Applicative translation with 10 selected lemmas
  • Monomorphizing translation with 10/100 selected lemmas
  • super with 10 selected lemmas
  • library_search
  • simp
  • refl

• Time limit of 30s for external provers + try_for 100000
  • longest total runtime is 125s
Success rate

% of non-refl theorems

- init
- logic
- tactic
- algebra
- order
- group theory
- geometry
- data
- field theory
- ring theory
- category theory
- set theory
- topology
- category
- linear algebra
- measure theory
- computability
- analysis
- number theory

success_hammer
Success rate, compared
Effect of monomorphization

The diagram shows the percentage of non-reflexive theorems for various directories, comparing monomorphization with and without. The x-axis represents different directories, and the y-axis shows the percentage of non-reflexive theorems. The blue bars represent monomorphization False, and the orange bars represent monomorphization True. The diagram indicates that monomorphization can significantly reduce the percentage of non-reflexive theorems in certain directories.
Robustness of reconstruction

- Number theory
- Set theory
- Logic
- Topology
- Measure theory
- Computability
- Data
- Tactic
- Analysis
- Linear algebra
- Group theory
- Initial
- Order
- Ring theory
- Category theory
- Algebra
- Category
- Geometry
- Field theory

\[ \text{additional success in } \% \]
\[ \text{extra_success_if_reconstruction_always_worked} \]
Lots of room for improvements—lemma selection

![Bar graph showing the percentage of non-refl theorems extracted from proofs for different directories.]

- lemmas_extracted_from_proof
  - False
  - True

Directory categories:
- init
- algebra
- order
- ring_theory
- tactic
- logic
- field_theory
- group_theory
- data
- set_theory
- geometry
- analysis
- measure_theory
- computability
- linear_algebra
- topology
- category_theory
- category
- number_theory

% of non-refl theorems

0 10 20 30 40 50
Introduction

Premise selection

Applicative translation to FOL

Monomorphizing translation to HOL

Empirical results

Demonstration

Conclusion
Introduction
Premise selection
Applicative translation to FOL
Monomorphizing translation to HOL
Empirical results
Demonstration

Conclusion
Conclusion

• Library design (such as type classes) has an effect on hammers
• Promising results, there is lots of room for improvement

• Next steps:
  • Improve premise selection
  • Copy-pastable tactic snippets
  • Increase cleverness of HOL translation
Lots of room for improvements—parsing failures

% of non-refl theorems

directory

prover_parsing_failure