CoqHammer: A General-Purpose Automated Reasoning Tool for Coq

Łukasz Czajka
University of Copenhagen

(joint work with Cezary Kaliszyk, University of Innsbruck)

24 January 2018

Interactive Proof in Type Theory

- Practical problem

Interactive Proof in Type Theory

- Practical problem
  - large parts of proofs are tedious

Interactive Proof in Type Theory

- Practical problem
  - large parts of proofs are tedious
- Automation for Interactive Proof
  - Proof search: intuition, firstorder,
  - Decision Procedures: congruence, fourier, ring, omega, SMTCoq, …

Interactive Proof in Type Theory

- Practical problem
  - large parts of proofs are tedious
- Automation for Interactive Proof
  - Proof search: intuition, firstorder,
  - Decision Procedures: congruence, fourier, ring, omega, SMTCoq, ...
- AI/ATP techniques: Hammers
  - MizAR for Mizar
  - Sledgehammer for Isabelle/HOL
  - HOL(y)Hammer for HOL Light and HOL4
  - **CoqHammer** for Coq

Hammers

- Hammer goal: provide efficient automated reasoning using facts from a large library.

Hammers

- Hammer goal: provide efficient automated reasoning using facts from a large library.
- Strong relevance filtering.

Hammers

- Hammer goal: provide efficient automated reasoning using facts from a large library.
- Strong relevance filtering.
- Usable library search “modulo simple reasoning”.

Hammers

- Hammer goal: provide efficient automated reasoning using facts from a large library.
- Strong relevance filtering.
- Usable library search “modulo simple reasoning”.
    - We may not know the name of the lemma we want to apply.

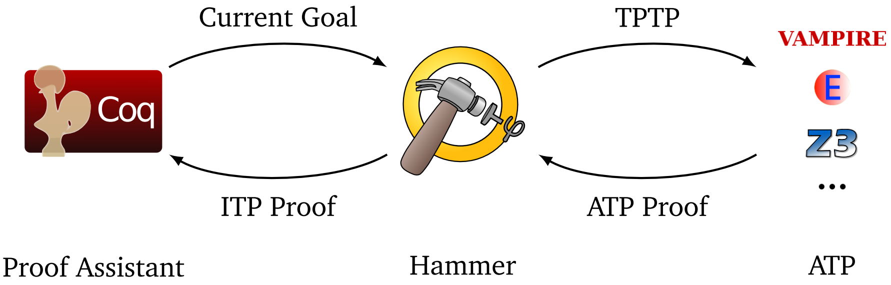
Hammers

- Hammer goal: provide efficient automated reasoning using facts from a large library.
- Strong relevance filtering.
- Usable library search “modulo simple reasoning”.
    - We may not know the name of the lemma we want to apply.
    - There may be many equivalent formulations of the lemma – which one is used in the library?

Hammers

- Hammer goal: provide efficient automated reasoning using facts from a large library.
- Strong relevance filtering.
- Usable library search "modulo simple reasoning".
  - We may not know the name of the lemma we want to apply.
  - There may be many equivalent formulations of the lemma – which one is used in the library?
  - The exact lemma may not exist in the library, but it may "trivially" follow from a few other lemmas in the library.

Hammer Overview



Hammers

Hammers work in three phases.

Hammers

Hammers work in three phases.

- Using machine-learning and AI techniques perform **premise-selection**: select about a few hundred to 1-2 thousand lemmas that are likely to be needed in the proof of the conjecture.

Hammers

Hammers work in three phases.

- Using machine-learning and AI techniques perform **premise-selection**: select about a few hundred to 1-2 thousand lemmas that are likely to be needed in the proof of the conjecture.
- **Translate** the selected lemmas, together with the conjecture, from the logic of the ITP to a format accepted by powerful external automated theorem provers (ATPs) – most commonly untyped first-order logic with equality.

Hammers

Hammers work in three phases.

- Using machine-learning and AI techniques perform **premise-selection**: select about a few hundred to 1-2 thousand lemmas that are likely to be needed in the proof of the conjecture.
- **Translate** the selected lemmas, together with the conjecture, from the logic of the ITP to a format accepted by powerful external automated theorem provers (ATPs) – most commonly untyped first-order logic with equality. Run the ATP(s) on the result of the translation.

Hammers

Hammers work in three phases.

- Using machine-learning and AI techniques perform **premise-selection**: select about a few hundred to 1-2 thousand lemmas that are likely to be needed in the proof of the conjecture.
- **Translate** the selected lemmas, together with the conjecture, from the logic of the ITP to a format accepted by powerful external automated theorem provers (ATPs) – most commonly untyped first-order logic with equality. Run the ATP(s) on the result of the translation.
- **Reprove** the conjecture in the logic of the ITP, using the information obtained in the ATP runs.

Hammers

Hammers work in three phases.

- Using machine-learning and AI techniques perform **premise-selection**: select about a few hundred to 1-2 thousand lemmas that are likely to be needed in the proof of the conjecture.
- **Translate** the selected lemmas, together with the conjecture, from the logic of the ITP to a format accepted by powerful external automated theorem provers (ATPs) – most commonly untyped first-order logic with equality. Run the ATP(s) on the result of the translation.
- **Reprove** the conjecture in the logic of the ITP, using the information obtained in the ATP runs. Typically, a list of (usually a few) lemmas needed by an ATP to prove the conjecture is obtained from an ATP run, and we try to reprove the goal from these lemmas.

Evaluations

Top-level goals:

- HOL(y)Hammer
  - Flyspeck text formalization: 47%
  - Similar results for HOL4
  - Slightly weaker for CakeML

Evaluations

Top-level goals:

- HOL(y)Hammer
  - Flyspeck text formalization: 47%
  - Similar results for HOL4
  - Slightly weaker for CakeML
- Sledgehammer
  - Probability theory: 40%
  - Term rewriting: 44%
  - Java threads: 59%

Evaluations

Top-level goals:

- HOL(y)Hammer
  - Flyspeck text formalization: 47%
  - Similar results for HOL4
  - Slightly weaker for CakeML
- Sledgehammer
  - Probability theory: 40%
  - Term rewriting: 44%
  - Java threads: 59%
- MizAR
  - Mizar Mathematical Library: 44%

Evaluations

Top-level goals:

- HOL(y)Hammer
  - Flyspeck text formalization: 47%
  - Similar results for HOL4
  - Slightly weaker for CakeML
- Sledgehammer
  - Probability theory: 40%
  - Term rewriting: 44%
  - Java threads: 59%
- MizAR
  - Mizar Mathematical Library: 44%
- **CoqHammer**
  - Coq standard library: 40%

CoqHammer demo

examples/imp.v

CoqHammer: premise selection

- Learning done each time the plugin is invoked
  (to include *all* accessible facts).

CoqHammer: premise selection

- Learning done each time the plugin is invoked
  (to include *all* accessible facts).
- Two machine-learning filters: k-NN and naive Bayes.

CoqHammer: premise selection

- Learning done each time the plugin is invoked
  (to include *all* accessible facts).
- Two machine-learning filters: k-NN and naive Bayes.
- Re-uses the HOLyHammer efficient implementation
  (also adapted by Sledgehammer).

Translation: target logic

Target logic: untyped FOL with equality.

Translation

Three functions $\mathscr{F}$, $\mathscr{G}$, and $\mathscr{C}$.

- $\mathscr{F}$: propositions $\rightarrow$ FOL formulas
  used for CIC_0 terms of type Prop.
- $\mathscr{G}$: types $\rightarrow$ guards
  used for CIC_0 terms of type Type.
- $\mathscr{C}$: all CIC_0 $\rightarrow$ FOL terms

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.
  - If $\Gamma \vdash t : \mathrm{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \rightarrow \mathscr{F}_{\Gamma,x:t}(s)$.
  - If $\Gamma \nvdash t : \mathrm{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x.\mathscr{G}_\Gamma(t, x) \rightarrow \mathscr{F}_{\Gamma,x:t}(s)$.

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.
  - If $\Gamma \vdash t : \mathrm{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \to \mathscr{F}_{\Gamma,x:t}(s)$.
  - If $\Gamma \not\vdash t : \mathrm{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x.\mathscr{G}_\Gamma(t,x) \to \mathscr{F}_{\Gamma,x:t}(s)$.
- The function $\mathscr{G}$ encodes types as guards and is used for terms of Coq which have type Type.

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.
  - If $\Gamma \vdash t : \mathrm{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \to \mathscr{F}_{\Gamma,x:t}(s)$.
  - If $\Gamma \nvdash t : \mathrm{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x.\mathscr{G}_\Gamma(t, x) \to \mathscr{F}_{\Gamma,x:t}(s)$.
- The function $\mathscr{G}$ encodes types as guards and is used for terms of Coq which have type Type.
  For instance, for a (closed) type $\tau = \Pi x : \alpha.\beta(x)$ we have

$$\mathscr{G}(\tau, f) = \forall x.\mathscr{G}(\alpha, x) \to \mathscr{G}(\beta(x), f\, x)$$

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.
  - If $\Gamma \vdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \to \mathscr{F}_{\Gamma,x:t}(s)$.
  - If $\Gamma \nvdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x.\mathscr{G}_\Gamma(t,x) \to \mathscr{F}_{\Gamma,x:t}(s)$.
- The function $\mathscr{G}$ encodes types as guards and is used for terms of Coq which have type Type.
  For instance, for a (closed) type $\tau = \Pi x : \alpha.\beta(x)$ we have

$$\mathscr{G}(\tau, f) = \forall x.\mathscr{G}(\alpha, x) \to \mathscr{G}(\beta(x), f\,x)$$

- The function $\mathscr{C}$ encodes Coq terms as FOL terms.

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.
  - If $\Gamma \vdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \to \mathscr{F}_{\Gamma, x:t}(s)$.
  - If $\Gamma \not\vdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x. \mathscr{G}_\Gamma(t, x) \to \mathscr{F}_{\Gamma, x:t}(s)$.
- The function $\mathscr{G}$ encodes types as guards and is used for terms of Coq which have type Type.
  For instance, for a (closed) type $\tau = \Pi x : \alpha.\beta(x)$ we have

$$\mathscr{G}(\tau, f) = \forall x. \mathscr{G}(\alpha, x) \to \mathscr{G}(\beta(x), f\,x)$$

- The function $\mathscr{C}$ encodes Coq terms as FOL terms.
  - $\mathscr{C}_\Gamma(ts)$ is equal to:

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.
  - If $\Gamma \vdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \to \mathscr{F}_{\Gamma,x:t}(s)$.
  - If $\Gamma \nvdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x.\mathscr{G}_\Gamma(t, x) \to \mathscr{F}_{\Gamma,x:t}(s)$.
- The function $\mathscr{G}$ encodes types as guards and is used for terms of Coq which have type Type.
  For instance, for a (closed) type $\tau = \Pi x : \alpha.\beta(x)$ we have

$$\mathscr{G}(\tau, f) = \forall x.\mathscr{G}(\alpha, x) \to \mathscr{G}(\beta(x), f\,x)$$

- The function $\mathscr{C}$ encodes Coq terms as FOL terms.
  - $\mathscr{C}_\Gamma(ts)$ is equal to:
    - ε if $\Gamma \vdash ts : \alpha : \text{Prop}$,

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.
  - If $\Gamma \vdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \rightarrow \mathscr{F}_{\Gamma,x:t}(s)$.
  - If $\Gamma \nvdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x.\mathscr{G}_\Gamma(t, x) \rightarrow \mathscr{F}_{\Gamma,x:t}(s)$.
- The function $\mathscr{G}$ encodes types as guards and is used for terms of Coq which have type Type.
  For instance, for a (closed) type $\tau = \Pi x : \alpha.\beta(x)$ we have

$$\mathscr{G}(\tau, f) = \forall x.\mathscr{G}(\alpha, x) \rightarrow \mathscr{G}(\beta(x), f\,x)$$

- The function $\mathscr{C}$ encodes Coq terms as FOL terms.
  - $\mathscr{C}_\Gamma(ts)$ is equal to:
    - ε if $\Gamma \vdash ts : \alpha : \text{Prop}$,
    - $\mathscr{C}_\Gamma(t)$ if $\Gamma \vdash s : \alpha : \text{Prop}$,

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.
  - If $\Gamma \vdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \to \mathscr{F}_{\Gamma,x:t}(s)$.
  - If $\Gamma \not\vdash t : \text{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x.\mathscr{G}_\Gamma(t,x) \to \mathscr{F}_{\Gamma,x:t}(s)$.
- The function $\mathscr{G}$ encodes types as guards and is used for terms of Coq which have type Type.
  For instance, for a (closed) type $\tau = \Pi x : \alpha.\beta(x)$ we have

$$\mathscr{G}(\tau, f) = \forall x.\mathscr{G}(\alpha, x) \to \mathscr{G}(\beta(x), f\,x)$$

- The function $\mathscr{C}$ encodes Coq terms as FOL terms.
  - $\mathscr{C}_\Gamma(ts)$ is equal to:
    - ε if $\Gamma \vdash ts : \alpha : \text{Prop}$,
    - $\mathscr{C}_\Gamma(t)$ if $\Gamma \vdash s : \alpha : \text{Prop}$,
    - $\mathscr{C}_\Gamma(t)\mathscr{C}_\Gamma(s)$ otherwise.

Translation

- The function $\mathscr{F}$ encodes propositions as FOL formulas and is used for terms of Coq having type Prop.
  - If $\Gamma \vdash t : \mathrm{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \to \mathscr{F}_{\Gamma,x:t}(s)$.
  - If $\Gamma \nvdash t : \mathrm{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x.\mathscr{G}_\Gamma(t,x) \to \mathscr{F}_{\Gamma,x:t}(s)$.
- The function $\mathscr{G}$ encodes types as guards and is used for terms of Coq which have type Type.
  For instance, for a (closed) type $\tau = \Pi x : \alpha.\beta(x)$ we have

$$\mathscr{G}(\tau, f) = \forall x.\mathscr{G}(\alpha, x) \to \mathscr{G}(\beta(x), f\,x)$$

- The function $\mathscr{C}$ encodes Coq terms as FOL terms.
  - $\mathscr{C}_\Gamma(ts)$ is equal to:
    - ε if $\Gamma \vdash ts : \alpha : \mathrm{Prop}$,
    - $\mathscr{C}_\Gamma(t)$ if $\Gamma \vdash s : \alpha : \mathrm{Prop}$,
    - $\mathscr{C}_\Gamma(t)\mathscr{C}_\Gamma(s)$ otherwise.
  - $\mathscr{C}_\Gamma(\lambda\vec{x} : \vec{t}.s) = F\vec{y}$ where s does not start with a lambda-abstraction any more, F is a fresh constant, $\vec{y} = \mathrm{FV}(\lambda\vec{x} : \vec{t}.s)$ and $\forall\vec{y}.\mathscr{F}_\Gamma(\forall\vec{x} : \vec{t}.F\vec{y}\vec{x} = s)$ is a new axiom.

ATP invocation

- We use Vampire, E prover, and Z3.

ATP invocation

- We use Vampire, E prover, and Z3.
- The provers may be run in parallel with different numbers of premises and premise selection methods.

Proof reconstruction

- Use dependencies from a successful ATP run.

Proof reconstruction

- Use dependencies from a successful ATP run.
- Do automatic proof search using different versions of our tactics (implemented in Ltac), with a fixed time limit for each.

Proof reconstruction

- Use dependencies from a successful ATP run.
- Do automatic proof search using different versions of our tactics (implemented in Ltac), with a fixed time limit for each.
- 85.2% of proofs reconstructed.

Overall hammer evaluation

All statements from the Coq standard libary

ATP success **50%**

- ATPs used: E, Z3, Vampire with 30 seconds time limit

Overall success **40.8%**

- 8 threads with different lemma selection, premises, provers, reconstruction

Conclusion

- Proof length already close to that of Isabelle/HOL.

Conclusion

- Proof length already close to that of Isabelle/HOL.
- Improvements needed for dependent types and boolean reflection.

Download

http://cl-informatik.uibk.ac.at/cek/coqhammer/