CoqHammer: Strong Automation for Program Verification

Łukasz Czajka
University of Copenhagen

Cezary Kaliszyk
University of Innsbruck

13 January 2018

Interactive Proof in Type Theory

- Why do we love it?

- Why do we hate it?

Interactive Proof in Type Theory

- Why do we love it?
  - The power we need
  - Successful projects
- Why do we hate it?
  - large parts of proofs are tedious

Interactive Proof in Type Theory

- Why do we love it?
  - The power we need
  - Successful projects
- Why do we hate it?
  - large parts of proofs are tedious
- Automation for Interactive Proof
  - Tableaux: Itaut, Tauto, Blast
  - Rewriting: Simp, Subst, HORewrite
  - Decision Procedures: Congruence Closure, Ring, Omega, SMTCoq, ...

Interactive Proof in Type Theory

- Why do we love it?
  - The power we need
  - Successful projects
- Why do we hate it?
  - large parts of proofs are tedious
- Automation for Interactive Proof
  - Tableaux: Itaut, Tauto, Blast
  - Rewriting: Simp, Subst, HORewrite
  - Decision Procedures: Congruence Closure, 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.

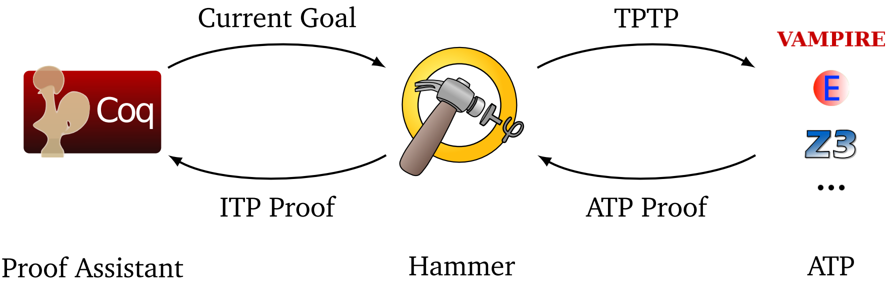
Hammers

- 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%
  - Programming languages: ?

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).

CoqHammer: premise selection – features

- Features $F(T)$ of theorem T
  - Constants occuring in the statement (type) of T.
  - Constant-constant and constant-variable pairs that share an edge in the parse tree.

CoqHammer: premise selection – features

```
Variable P : nat -> Prop.
T = forall k l, between k l -> k <= l.
```

CoqHammer: premise selection – features

```
Variable P : nat -> Prop.
T = forall k l, between k l -> k <= l.
```

$F(T)$ = { "Between.Between.between",
"Between.Between.between-X",
"Coq.Init.Datatypes.nat", "Coq.Init.Peano.le",
"Coq.Init.Peano.le-X"}

CoqHammer: premise selection – dependencies

- Dependencies $D(T)$ of theorem (resp. definition) T
    - Constants occuring in the proof term (resp. unfolding) of T.

CoqHammer: premise selection – dependencies

Proof term of T:

```
fun (k l : nat) (H : between P k l) =>
between_ind P k (fun l0 : nat => k <= l0) (le_n k)
  (fun (l0 : nat) (_ : between P k l0)
  (IHbetween : k <= l0) (_ : P l0) =>
   le_S k l0 IHbetween) l H
```

CoqHammer: premise selection – dependencies

Proof term of T:

```
fun (k l : nat) (H : between P k l) =>
between_ind P k (fun l0 : nat => k <= l0) (le_n k)
  (fun (l0 : nat) (_ : between P k l0)
  (IHbetween : k <= l0) (_ : P l0) =>
   le_S k l0 IHbetween) l H
```

$D(T) \;=\{$ `"Between.Between.between",`
`"Between.Between.between_ind",`
`"Coq.Init.Datatypes.nat", "Coq.Init.Peano.le",`
`"Coq.Init.Peano.le_S", "Coq.Init.Peano.le_n",`
`"P"`$\}$

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 $\mathscr{F}_\Gamma(\Pi x : t.s) = \mathscr{F}_\Gamma(t) \to \mathscr{F}_{\Gamma, x:t}(s)$.
  - If $\Gamma \nvdash t : \mathrm{Prop}$ then $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x. \mathscr{G}_\Gamma(t, x) \to \mathscr{F}_{\Gamma, x:t}(s)$.

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.

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 $\Gamma \vdash t : \mathrm{Prop}$ 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.

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:

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 $\Gamma \vdash t : \mathrm{Prop}$ 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 $\Gamma \vdash s : \alpha : \mathrm{Prop}$,

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 $\Gamma \vdash ts : \alpha : \text{Prop}$,
    - $\mathscr{C}_\Gamma(t)$ 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 $\Gamma \vdash t : \text{Prop}$ 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 $\Gamma \vdash ts : \alpha : \text{Prop}$,
    - $\mathscr{C}_\Gamma(t)$ if $\Gamma \vdash s : \alpha : \text{Prop}$,
    - $\mathscr{C}_\Gamma(t)\mathscr{C}_\Gamma(s)$ 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, $\vec{y} = \text{FV}(\lambda\vec{x} : \vec{t}.s)$ and $\forall\vec{y}.\mathscr{F}_\Gamma(\forall\vec{x} : \vec{t}.F\vec{y}\vec{x} = s)$ is a new axiom.

Translation: translating inductive declarations

For inductive types:

- Translate the typing of each constructor (using the $\mathscr{G}$ function).

Translation: translating inductive declarations

For inductive types:

- Translate the typing of each constructor (using the $\mathscr{G}$ function).
- Add axioms stating injectivity of constructors, axioms stating non-equality of different constructors, and the "inversion" axioms for elements of the inductive type.

Translation: translating inductive declarations

For inductive types:

- Translate the typing of each constructor (using the $\mathscr{G}$ function).
- Add axioms stating injectivity of constructors, axioms stating non-equality of different constructors, and the "inversion" axioms for elements of the inductive type.
- Translate the typing of the inductive definition.

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.

Proof search

$$\frac{\texttt{eauto}[\Gamma \vdash G]}{\Gamma \vdash G} \quad \frac{\texttt{congruence}[\Gamma \vdash G]}{\Gamma \vdash G} \quad \frac{\texttt{constructor}[\Gamma \vdash G]}{\Gamma \vdash G} \quad \frac{\texttt{easy}[\Gamma \vdash G]}{\Gamma \vdash G}$$

Leaf tactics (b)

$$\frac{\Gamma; A[?e_1/x_1,\ldots,?e_n/x_n] \vdash G \quad \Gamma; B[?e_1/x_1,\ldots,?e_n/x_n] \vdash G}{\Gamma; \forall x_1 \ldots x_n, A \vee B \vdash G}\ (\text{orsplit}) \quad \frac{\Gamma; A[?e_1/x_1,\ldots,?e_n/x_n] \vdash G \quad y \text{ fresh}}{\Gamma; \forall x_1 \ldots x_n, \exists y, A \vdash G}\ (\text{exsimpl})$$

Final splitting (e)

$$\frac{\texttt{destruct}(t)[\Gamma \vdash C[\texttt{match}\ t\ \texttt{with}\ b]]}{\Gamma \vdash C[\texttt{match}\ t\ \texttt{with}\ b]} \quad \frac{\texttt{destruct}(t)[\Gamma; C[\texttt{match}\ t\ \texttt{with}\ b] \vdash G]}{\Gamma; C[\texttt{match}\ t\ \texttt{with}\ b] \vdash G} \quad \frac{\Gamma \vdash G_1 \quad \Gamma \vdash G_2}{\Gamma \vdash G_1 \wedge G_2}$$

Splitting (e)

$$\frac{\Gamma; A; B \vdash G}{\Gamma; A; A \to B \vdash G} \quad \frac{\Gamma; \forall x_1 \ldots x_n, A; \forall x_1 \ldots x_n, B \vdash G}{\Gamma; \forall x_1 \ldots x_n, A \wedge B \vdash G}$$

$$\frac{\Gamma; \forall x_1 \ldots x_n, A \to B \to C \vdash G}{\Gamma; \forall x_1 \ldots x_n, A \wedge B \to C \vdash G} \quad \frac{\Gamma; \forall x_1 \ldots x_n, A \to C; \forall x_1 \ldots x_n, B \to C \vdash G}{\Gamma; \forall x_1 \ldots x_n, A \vee B \to C \vdash G} \quad \frac{\Gamma; A \vdash G \quad x \text{ fresh}}{\Gamma; \exists x, A \vdash G} \quad \frac{\Gamma; A \vdash G}{\Gamma; A; A \vdash G}$$

Hypothesis simplification (e)

$$\frac{\Gamma; x : A \vdash B}{\Gamma \vdash \forall x : A, B}\ (\text{intro})$$

Introduction (e)

$$\frac{}{\Gamma; \text{False} \vdash G} \quad \frac{}{\Gamma; G \vdash G} \quad \frac{\Gamma \vdash G[?e/x]}{\Gamma \vdash \exists x, G} \quad \frac{\Gamma \vdash G_1}{\Gamma \vdash G_1 \vee G_2} \quad \frac{\Gamma \vdash G_2}{\Gamma \vdash G_1 \vee G_2}$$

$$\frac{\texttt{yapply}(H)[\Gamma; H : A \vdash P]}{\Gamma; H : A \vdash P}\ (\text{apply}) \quad \frac{\texttt{yrewrite}(H)[\Gamma; H : \forall x_1 \ldots x_n, A = B \vdash G]}{\Gamma; H : \forall x_1 \ldots x_n, A = B \vdash G}\ (\text{rewrite})$$

$$\frac{\Gamma; \forall x_{k+1} \ldots x_n, \exists x, A[?e_1/x_1,\ldots,?e_k/x_k] \vdash G}{\Gamma; \forall x_1 \ldots x_k \ldots x_n, \exists x, A \vdash G}\ (\text{exinst}) \quad \frac{\Gamma; T'_{j_1},\ldots,T'_{j_{m:n}}, A'_1 \vdash G \quad \Gamma; T'_{j_1},\ldots,T'_{j_{m:n}}, A'_2 \vdash G \quad \Gamma; T'_{j_1},\ldots,T'_{j_{m:k-1}} \vdash T'_k}{\Gamma; \forall (x_1 : T_1) \ldots (x_n : T_n), A_1 \vee A_2 \vdash G}\ (\text{orinst})$$

Proof search (b)

$$\frac{\Gamma; \forall x_1 \ldots x_n, \text{False} \vdash \text{False}}{\Gamma; \forall x_1 \ldots x_n, \text{False} \vdash G} \quad \frac{\texttt{inversion}(H)[\Gamma; H : P \vdash G]}{\Gamma; H : P \vdash G}\ (\text{invert}) \quad \frac{\texttt{destruct}(x)[\Gamma; x : T \vdash G]}{\Gamma; x : T \vdash G}\ (\text{destruct})$$

Initial proof search (b)

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

Works well for program semantics formalizations.

Conclusion

- CoqHammer optimized for programming language formalization.

Conclusion

- CoqHammer optimized for programming language formalization.
- Proof length already close to that of Isabelle/HOL.

Conclusion

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