Components of a Hammer for Type Theory

Goal Translation and Proof Reconstruction

Łukasz Czajka   Cezary Kaliszyk

University of Innsbruck

May 24, 2016

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 today
- Why do we hate it?
  - ITPs are stupid
  - large parts of proofs are tedious

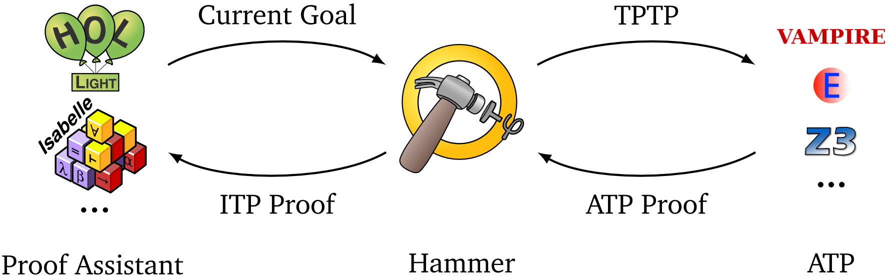
Interactive Proof in Type Theory

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

Interactive Proof in Type Theory

- Why do we love it?
  - The power we need
  - Successful projects today
- Why do we hate it?
  - ITPs are stupid
  - large parts of proofs are tedious
- Automation for Interactive Proof
  - Tableaux: Itaut, Tauto, Blast
  - Rewriting: Simp, Subst, HORewrite
  - Decision Procedures: Congruence Closure, Ring, Omega, Cooper, ...
- AI/ATP techniques: Hammers
  - MizAR for Mizar
  - Sledgehammer for Isabelle/HOL
  - HOL(y)Hammer for HOL Light and HOL4

Hammer Overview



Evaluations

Top-level goals:

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

More for subgoals

For Type Theory?

Premise selection

- Features
- Machine Learning

Encoding CoC and variants in formalisms of ATPs

- Soundness? Completeness? Efficiency!
- This talk

Reconstruction: Get an ITP proof

- Extract information from the ATP proof
- Redo the proof

Translation

Target logic

Target logic: untyped FOL with equality.

Translation

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

Translation

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

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

Translation

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

- 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 $\mathscr{F}_\Gamma(\Pi x : t.s) = \forall x.\mathscr{G}_\Gamma(t, x) \rightarrow \mathscr{F}_{\Gamma, x:t}(s)$.

Translation

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

- 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

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

- 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

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

Translation

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

- 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

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

- 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:

Translation

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

- 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 $\Gamma \vdash ts : \alpha : \mathrm{Prop}$,

Translation

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

- 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.
  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

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

- 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.

Translation

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

- 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.

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 $\mathcal{G}$ 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.

Translation

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.
- Translate induction principles and recursor definitions.

Proof reconstruction

- From an ATP run we obtain a list of FOL axioms that the ATP needed in the proof.

Proof reconstruction

- From an ATP run we obtain a list of FOL axioms that the ATP needed in the proof.
- Extract from the FOL axiom names the names of the original Coq lemmas and constructors, and add them to the context.

Proof reconstruction

- From an ATP run we obtain a list of FOL axioms that the ATP needed in the proof.
- Extract from the FOL axiom names the names of the original Coq lemmas and constructors, and add them to the context.
- Extract from the FOL axiom names the names of definitions used and try unfolding them (depending on some heuristics).

Proof reconstruction

- From an ATP run we obtain a list of FOL axioms that the ATP needed in the proof.
- Extract from the FOL axiom names the names of the original Coq lemmas and constructors, and add them to the context.
- Extract from the FOL axiom names the names of definitions used and try unfolding them (depending on some heuristics).
- Do automatic proof search using our tactic `yreconstr`.

Proof search

- Essentially `eauto`-type proof search.

Proof search

- Essentially `eauto`-type proof search.
- In other words: search for η-long normal forms (taking the permutative conversions into account).

Proof search

- Essentially `eauto`-type proof search.
- In other words: search for η-long normal forms (taking the permutative conversions into account).
- When doing `intro` try simplifying the introduced hypothesis, heurstically rewriting it with other hypotheses, and doing some simple forward reasoning.

Proof search

- Essentially `eauto`-type proof search.
- In other words: search for η-long normal forms (taking the permutative conversions into account).
- When doing `intro` try simplifying the introduced hypothesis, heurstically rewriting it with other hypotheses, and doing some simple forward reasoning.
- In the proof search also try rewriting with hypotheses instead of only applying them.

Proof search

- Essentially `eauto`-type proof search.
- In other words: search for η-long normal forms (taking the permutative conversions into account).
- When doing `intro` try simplifying the introduced hypothesis, heurstically rewriting it with other hypotheses, and doing some simple forward reasoning.
- In the proof search also try rewriting with hypotheses instead of only applying them.
- When applying a hypothesis try to unify the goal with the hypothesis target modulo some simple equational reasoning.

Proof search

- Essentially `eauto`-type proof search.
- In other words: search for η-long normal forms (taking the permutative conversions into account).
- When doing `intro` try simplifying the introduced hypothesis, heurstically rewriting it with other hypotheses, and doing some simple forward reasoning.
- In the proof search also try rewriting with hypotheses instead of only applying them.
- When applying a hypothesis try to unify the goal with the hypothesis target modulo some simple equational reasoning.
- If a subterm of the form match x with ... occurs in the goal or in one of the hypotheses, then destruct x.

Proof search

- Essentially `eauto`-type proof search.
- In other words: search for η-long normal forms (taking the permutative conversions into account).
- When doing `intro` try simplifying the introduced hypothesis, heurstically rewriting it with other hypotheses, and doing some simple forward reasoning.
- In the proof search also try rewriting with hypotheses instead of only applying them.
- When applying a hypothesis try to unify the goal with the hypothesis target modulo some simple equational reasoning.
- If a subterm of the form match x with . . . occurs in the goal or in one of the hypotheses, then destruct x.
- Use an `isolve` tactic at the leaves of the search tree: a combination of Coq's `congruence`, `subst`, `easy`, `eauto` tactics, some hypotheses simplification and goal splitting.

Experimental evaluation

- We evaluated the translation and proof reconstruction components on the Coq standard library.

Experimental evaluation

- We evaluated the translation and proof reconstruction components on the Coq standard library.
- ATPs used: Z3, Vampire, E.

Experimental evaluation

- We evaluated the translation and proof reconstruction components on the Coq standard library.
- ATPs used: Z3, Vampire, E.
- Success rate of the ATPs on translated problems: about 35%.

Experimental evaluation

- We evaluated the translation and proof reconstruction components on the Coq standard library.
- ATPs used: Z3, Vampire, E.
- Success rate of the ATPs on translated problems: about 35%.

Prover	Solved%	Solved	Sum%	Sum	Unique
Vampire	32.9	6839	32.9	6839	855
Z3	27.6	5734	34.9	7265	390
E Prover	25.8	5376	35.3	7337	72
any	35.3	7337	35.3	7337	

Table 1: Results of the experimental evaluation on the 20803 FOL problems generated from the propositions in the Coq standard library.

Experimental evaluation

- Reconstruction success rate: 90%.

Experimental evaluation

- Reconstruction success rate: 90%.

  But many of the Coq problems recreated from ATP runs are “easy”:

Experimental evaluation

- Reconstruction success rate: 90%.

  But many of the Coq problems recreated from ATP runs are “easy”:

  - about 50% provable using `intuition`, `congruence`, `auto` and hypotheses simplification.

Experimental evaluation

- Reconstruction success rate: 90%.

  But many of the Coq problems recreated from ATP runs are “easy”:

  - about 50% provable using `intuition`, `congruence`, `auto` and hypotheses simplification.
  - about 70% provable using the above plus `isolve` and exhaustive search up to depth 2 using `eapply` and `erewrite`.

Experimental evaluation

- Reconstruction success rate: 90%.

  But many of the Coq problems recreated from ATP runs are “easy”:

  - about 50% provable using `intuition`, `congruence`, `auto` and hypotheses simplification.
  - about 70% provable using the above plus `isolve` and exhaustive search up to depth 2 using `eapply` and `erewrite`.
  - about 70% provable using `firstorder isolve` provided that generic equality axioms are added to the context.

Experimental evaluation

Tactic	Time	Solved%	Solved
`yreconstr`	1s	83.1	6097
`yreconstr`	2s	85.8	6296
`yreconstr`	5s	87.5	6421
`yreconstr`	10s	88.1	6466
`yreconstr`	15s	88.2	6473
`simple`	1s	50.1	3674
`firstorder'`	10s	69.6	5103
`jprover`	10s	56.1	4114
any		90.1	6609

Table 2: Results of the evaluation of proof reconstruction on the 7337 problems solved by the ATPs.

Conclusion

- Provided missing components of a hammer for type theory
- Efficient encoding in FOL
  - Able to automatically prove 35% of Coq's standard library
- Simple reconstruction
  - 90% of the ATP-found proofs can be rebuilt in Coq
- Other libraries?
  - Mathematical Components / SS-Reflect where different automation?
  - Libraries of Matita, Lean, ...?
- Optimize, optimize, optimize!
  - Learning
  - Translation
  - Reconstruction