Goal Translation for a Hammer for Coq (Extended Abstract)

Łukasz Czajka  
lukasz.czajka@uibk.ac.at  
University of Innsbruck, Austria

Cezary Kaliszyk  
cezary.kaliszyk@uibk.ac.at  
University of Innsbruck, Austria

Hammers are tools that provide general purpose automation for formal proof assistants. Despite the gaining popularity of the more advanced versions of type theory, there are no hammers for such systems. We present an extension of the various hammer components to type theory: (i) a translation of a significant part of the Coq logic into the format of automated proof systems; (ii) a proof reconstruction mechanism based on a Ben-Yelles-type algorithm combined with limited rewriting, congruence closure and a first-order generalization of the left rules of Dyckhoff’s system LJT.

1 Introduction

Justifying small proof steps is usually a significant part of the process of formalizing proofs in an interactive theorem proving (ITP), or proof assistant, system. Many of such goals would be considered trivial by mathematicians. Still, state-of-the-art ITPs require the user to spend an important part of the formalization effort on them. The main points that constitute this effort are usually library search, minor transformations on the already proved theorems (such as reordering assumptions or reasoning modulo associativity-commutativity), as well as combining a small number of simple known lemmas. To reduce this effort various automation techniques have been conceived, including techniques from automated reasoning and domain specific decision procedures. The strongest general propose automation technique, available for various interactive theorem provers today is provided by “hammers” [10].

Hammers are proof assistant tools that employ external automated theorem provers (ATPs) in order to automatically find proofs of user given conjectures. There are three main components of a hammer:

- Lemma selection (also called relevance filtering or premise selection) that heuristically chooses a subset of the accessible lemmas that are likely useful for the given conjecture.
- Translation (encoding) of the user given conjecture together with the selected lemmas to the logics and input formats of automated theorem provers (ATPs). The focus is usually on first-order logic as the majority of the most efficient ATPs today support this foundation. The automated systems are in turn used to either find an ATP proof or just further narrow down the subset of lemmas to precisely those that are necessary in the proof.
- Proof reconstruction, which uses the obtained information from the successful ATP run, to reprove the lemma in the logic of the proof assistant.

Robust hammers exist for proof assistants based on higher-order logic (Sledgehammer [27] for Isabelle/HOL [33], HOLyHammer [20] for HOL Light [13] and HOL4 [31]) or dependently typed set theory (MizAR [21] for Mizar [7, 34, 6]). The general-purpose automation provided by the most advanced hammers is able to solve 40–50% of the top-level goals in various developments [10], as well as more than 70% of the user-visible subgoals [11], and as such has been found very useful in various proof developments [17].

Despite the gaining popularity of the more advanced versions of type theory, implemented by systems such as Agda [12], Coq [8], Lean [25], and Matita [4], there are no hammers for such systems. The
construction of such a tool has so far been hindered by the lack of a usable encoding component, as well as by comparatively weak proof reconstruction.

For the proof assistants whose logics are based on the Calculus of Constructions and its extensions, the existing encodings in first-order logic so far cover only limited fragments of the source logic \([1, 32, 9]\). Why3 \([16]\) provides a translation from its own logic \([15]\) (which is a subset of the Coq logic, including features like rank-1 polymorphism, algebraic data types, recursive functions and inductive predicates) to the format of various first-order provers (in fact Why3 has been initially used as a translation backend for HOLyHammer). Recently, an encoding of the dependently typed higher-order logic of \(F^*\) into first-order logic has also been developed \([2]\).

The built-in HOL automation is able to reconstruct the majority of the automatically found proofs using either internal proof search \([19]\) or source-level reconstruction. The internal proof search mechanisms provided in Coq, such as the \texttt{firstorder} tactic \([13]\), have been insufficient for this purpose so far. Matita’s ordered paramodulation \([5]\) is able to reconstruct many goals with up to two or three premises, and the congruence-closure based internal automation techniques in Lean \([24]\) are also promising.

The SMTCoq \([3]\) project has developed an approach to use external SAT and SMT solvers and verify their proof witnesses. Small checkers are implemented using reflection for parts of the SAT and SMT proof reconstruction, such as one for CNF computation and one for congruence closure. The procedure is able to handle Coq goals in the subset of the logic that corresponds to the logics of the input systems.

**Contributions.** We present our recently developed proof advice components for type theory and systems based on it. We first introduce an encoding of the Calculus of Inductive Constructions, including the additional logical constructions introduced by the Coq system, in untyped first-order logic with equality. We implement the translation and evaluate it experimentally on the standard library of the Coq proof assistant. We advocate that the encoding is sufficient for a hammer system for Coq: the success rates are comparable to those demonstrated by early hammer systems for Isabelle/HOL and Mizar, while the dependencies used in the ATP proofs are most often sufficient to prove the original theorems. Strictly speaking, our translation is neither sound nor complete. However, our experiments suggest that the encoding is “sound enough” to be usable. Moreover, we believe that a “core” version of the translation is sound and we are currently working on a proof of this fact.

Secondly, we present a proof reconstruction mechanism based on a Ben-Yelles-type procedure combined with a first-order generalization of the left rules of Dyckhoff’s LJT, congruence closure and heuristic rewriting. With this still preliminary proof search procedure we are able to reprove almost 90% of the problems solved by the ATPs, using the dependencies extracted from the ATP output.

## 2 Translation

In this section we introduce an encoding of (a close approximation of) the Calculus of Inductive Constructions into untyped first-order logic with equality. The encoding should be a practical one, which implies that its general theoretical soundness is not the main focus, i.e., of course the translation needs to be “sound enough” to be usable, but it is more important that the encoding is efficient enough to provide practically useful information about the necessary proof dependencies. In particular, the encoding needs to be shallow, meaning that Coq terms of type \(\text{Prop}\) are translated directly to corresponding first-order formulas. Our translation is in fact unsound, e.g., it assumes proof irrelevance and ignores certain universe constraints. However, we believe that under the assumption of proof irrelevance a “core” version of the translation is sound, and we are currently working on a proof.
Below we present a variant of the translation for a fragment of the logic of Coq. The intention here is to provide a general idea, but not to describe the encoding in detail. In the first-order language we assume a unary predicate $P$, a binary predicate $T$ and a binary function symbol $@$. Usually, we write $ts$ instead of $@ (t,s)$.

For the sake of efficiency, terms of type Prop are encoded directly as FOL formulas using a function $\mathcal{F}$. Terms that have type Type but not Prop are encoded using a function $\mathcal{G}$ as guards which essentially specify what it means for an object to have the given type. For instance, $\forall f : \tau . \varphi$ where $\tau = \Pi x : \alpha . \beta$ is translated to $\forall f . \mathcal{G} (\tau , f) \rightarrow \mathcal{F} (\varphi)$ where $\mathcal{G} (\tau , f) = \forall x . \mathcal{G} (\alpha , x) \rightarrow \mathcal{G} (\beta , f x)$. So $\mathcal{G} (\tau , f)$ says that an object $f$ has type $\tau = \Pi x : \alpha . \beta$ if for any object $x$ of type $\alpha$, the application $f x$ has type $\beta$.

Function $\mathcal{G}$ encoding propositions as FOL formulas is defined by:

- If $\Gamma \vdash f : \tau \rightarrow \sigma$ then $\mathcal{G} (f) = \mathcal{F} (\forall x . \mathcal{G} (x , \sigma))$.
- Otherwise, if none of the above apply, $\mathcal{G} (f) = \mathcal{F} (\exists x . \mathcal{G} (x , \tau))$.

Function $\mathcal{G}$ encoding types as guards is defined by:

- If $\Gamma \vdash \Pi x : t_1 , t_2 , \tau$ then $\mathcal{G} (\Pi x : t_1 , t_2 , \tau) = \mathcal{F} (\forall x . \mathcal{G} (x , \tau))$.
- Otherwise, when $\Gamma \vdash \tau : \sigma$ then $\mathcal{G} (\tau) = \mathcal{F} (\forall x . \mathcal{G} (x , \sigma))$.

Function $\mathcal{G}$ encoding terms as FOL terms is defined by:

- If $\Gamma \vdash b$ then $\mathcal{G} (b) = \mathcal{F} (b)$.
- Otherwise, if none of the above apply, $\mathcal{G} (b) = \mathcal{F} (\exists x . \mathcal{G} (x , b))$.

For instance, $\mathcal{G} (\lambda x . \psi)$ is translated to $\mathcal{F} (\lambda x . \mathcal{G} (\psi))$ where $\mathcal{G} (\psi)$ is a new axiom.

The following is a new axiom:

\[
\forall \gamma_1 . \mathcal{G}_1 (\forall y_1 . \mathcal{F}_1 (\exists z_1 : [\beta_1 , \beta_2 , \beta_3 , \beta_4 , \beta_5 , \beta_6 , \beta_7 , \beta_8 ] . \forall x_1 : [\tau_1 , s_1 , \ldots , \tau_k , s_k ] . \forall y_2 . [\gamma_1 \tau_1 \gamma_2 s_1 ] . (1 z_1 : [\beta_1 , \beta_2 , \beta_3 , \beta_4 , \beta_5 , \beta_6 , \beta_7 , \beta_8 ] . \forall x_1 : [\tau_1 , s_1 , \ldots , \tau_k , s_k ] . \forall y_2 . [\gamma_1 \tau_1 \gamma_2 s_1 ] . ) = 1 y_1 y_2 \text{ for a fresh constant } F \text{ where }
\]

\[
\begin{align*}
\Gamma_2 & = y_2 : [\beta_2] \mathcal{F} (\Gamma_1) , \\
\Gamma_1 & = y_1 : [\beta_1] \mathcal{F} (\Gamma_0) , \\
\Gamma_0 & = \gamma_1 \Gamma_1 : [\beta_0] \mathcal{G}_0 (\forall x_1 : [\tau_1 , s_1 , \ldots , \tau_k , s_k ] . \forall y_1 , y_2 . [\gamma_1 \tau_1 \gamma_2 s_1 ] . ) , \\
\gamma_1 & = \Pi \gamma_2 : [\beta_0] \mathcal{G}_0 (\forall x_1 : [\tau_1 , s_1 , \ldots , \tau_k , s_k ] . \forall y_1 , y_2 . [\gamma_1 \tau_1 \gamma_2 s_1 ] . ) , \\
\end{align*}
\]

- the following is a new axiom:

\[
\forall y_1 . \mathcal{G}_1 (\forall y_2 : [\beta_2] . ( \exists z_1 : [\beta_1] . \exists x_1 : [\tau_1 , t = c_1 z_1 x_1 \land F y_1 y_2 = s_1 ] ) \lor \ldots \lor ( \exists z_k : [\beta_k] . \exists x_k : [\tau_k , t = c_k z_k x_k \land F y_1 y_2 = s_k ] ) )
\]
Here \( t \) is the term matched on, the type of \( t \) has the form \( c \overline{\vec{u}} \), the integer \( n \) denotes the number of parameters (which is the length of \( \vec{p} \)), the type \( \tau|\overline{\vec{a},t/x} \) is the return type, i.e., the type of the whole case expression, \( \vec{a} \cap \text{FV}(\vec{p}) = \emptyset \), and \( s|\overline{\vec{v},\vec{x}} \) is the value of the case expression if the value of \( t \) is \( c_1 \overline{\vec{v}} \). The free variable context \( \text{FC}(\Gamma;i) \) is defined inductively: \( \text{FC}(\emptyset;i) = \emptyset \); \( \text{FC}(\Gamma,x: \tau;i) = \text{FC}(\Gamma;Ax: \tau.i), x: \tau \) if \( x \in \text{FV}(i) \); and \( \text{FC}(\Gamma,x: \tau;i) = \text{FC}(\Gamma;i) \) if \( x \notin \text{FV}(i) \).

In the data exported from Coq there are three types of declarations: definitions, typing declarations and inductive declarations. We briefly describe how all of them are translated.

A definition \( c = t: \tau : \kappa \) is translated as follows.

- If \( \kappa = \text{Prop} \) then add \( \mathcal{F}(\tau) \) as a new axiom with label \( c \).
- If \( \kappa \neq \text{Prop} \) then
  - add \( \mathcal{G}(\tau,c) \) as a new axiom,
  - if \( \tau = \text{Prop} \) then add \( c \leftrightarrow \mathcal{F}(t) \) as a new axiom with label \( c \),
  - if \( \tau = \text{Set} \) or \( \tau = \text{Type} \) then add \( \forall f.f \leftrightarrow \mathcal{G}(t,f) \) as a new axiom with label \( c \),
  - if \( \tau \notin \{\text{Prop},\text{Set},\text{Type}\} \) then add the equation \( c = \mathcal{G}(t) \) as a new axiom with label \( c \).

A typing declaration \( c : \tau : \kappa \) is translated as follows.

- If \( \kappa = \text{Prop} \) then add \( \mathcal{F}(\tau) \) as a new axiom with label \( c \).
- If \( \kappa \neq \text{Prop} \) then add \( \mathcal{G}(\tau,c) \) as a new axiom with label \( c \).

An inductive declaration \( I(c: \tau : \kappa := c_1 : \tau_1 : \kappa_1, \ldots, c_n : \tau_n : \kappa_n) \) is translated as follows.

- Translate the typing declaration \( c : \tau : \kappa \).
- Translate each typing declaration \( c_i : \tau_i : \kappa \) for \( i = 1, \ldots, n \).
- Add axioms stating injectivity of constructors, axioms stating non-equality of different constructors, and the “inversion” axioms for elements of the inductive type.

For inductive types also induction principles and recursor definitions are translated.

The above only gives a general outline of the translation. In practice, we make a number of optimisations, e.g., the arity optimisation by Meng and Paulson [23], or translating fully applied functions with target type Prop directly to first-order predicates.

### 3 Reconstruction

We report on our work on proof reconstruction. We evaluate the Coq internal reconstruction mechanisms including \texttt{tauto} and \texttt{firstorder} [13] on the original proof dependencies and on the ATP found proofs, which are in certain cases more precise. In particular \texttt{firstorder} seems insufficient for finding proofs for problems created using the advice obtained from the ATP runs. This is partly caused by the fact that it does not fully axiomatize equality, but even on problems which require only purely logical first-order reasoning its running time is sometimes unacceptable.

The formulas that we attempt to reprove usually belong to fragments of intuitionistic logic low in the Mints hierarchy [29]. Most of proved theorems follow by combining a few known lemmas. This raises a possibility of devising an automated proof procedure optimized for these fragments of intuitionistic logic, and for the usage of the advice obtained from the ATP runs. We implemented a preliminary version of a Ben-Yelles-type procedure (essentially \texttt{eauto}-type proof search with a looping check) augmented with
Table 1: Results of the experimental evaluation on the 20803 FOL problems generated from the propositions in the Coq standard library.

<table>
<thead>
<tr>
<th>Prover</th>
<th>Solved%</th>
<th>Solved</th>
<th>Sum%</th>
<th>Sum</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vampire</td>
<td>32.9</td>
<td>6839</td>
<td>32.9</td>
<td>6839</td>
<td>855</td>
</tr>
<tr>
<td>Z3</td>
<td>27.6</td>
<td>5734</td>
<td>34.9</td>
<td>7265</td>
<td>390</td>
</tr>
<tr>
<td>E Prover</td>
<td>25.8</td>
<td>5376</td>
<td>35.3</td>
<td>7337</td>
<td>72</td>
</tr>
<tr>
<td>any</td>
<td>35.3</td>
<td>7337</td>
<td>35.3</td>
<td>7337</td>
<td></td>
</tr>
</tbody>
</table>

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

4 Evaluation

We evaluated our translation on the problems generated from all declarations of terms of type Prop in the Coq standard library of Coq version 8.5. We used the following classical ATPs: E Prover version 1.9 [30], Vampire version 4.0 [22] and Z3 version 4.0 [26]. The methodology was to measure the number of theorems that the ATP could reprove from their extended dependencies within a time limit of 30 s for each problem. The extended dependencies of a theorem are obtained by taking all constants occurring in the proof term of the theorem in Coq standard library, and recursively taking all constants occurring in the types and non-proof definitions of any dependencies extracted so far. Because of the use of extended dependencies, the average number of generated FOL axioms for a problem is 193. We limited the recursive extraction of extended dependencies to depth 2.

The evaluation was performed on a 48-core server with 2.2 GHz AMD Opteron CPUs and 320 GB RAM. Each problem was always assigned one CPU core. Table 1 shows the results of our evaluation. The column “Solved%” denotes the percentage (rounded to the first decimal place) of the problems solved by a given prover, and “Solved” the number of problems solved out of the total number of 20803 problems. The column “Sum%” denotes the percentage, and “Sum” the total number, of problems solved by the prover or any of the provers listed above it. The column “Unique” denotes the number of problems the given prover solved but no other prover could solve.

We also evaluated various proof reconstruction mechanisms on the problems originating from ATP...
<table>
<thead>
<tr>
<th>Tactic</th>
<th>Time</th>
<th>Solved%</th>
<th>Solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>yreconstr0</td>
<td>10s</td>
<td>26.8</td>
<td>1965</td>
</tr>
<tr>
<td>yreconstr</td>
<td>1s</td>
<td>83.1</td>
<td>6097</td>
</tr>
<tr>
<td>yreconstr</td>
<td>2s</td>
<td>85.8</td>
<td>6296</td>
</tr>
<tr>
<td>yreconstr</td>
<td>5s</td>
<td>87.5</td>
<td>6421</td>
</tr>
<tr>
<td>yreconstr</td>
<td>10s</td>
<td>88.1</td>
<td>6466</td>
</tr>
<tr>
<td>yreconstr</td>
<td>15s</td>
<td>88.2</td>
<td>6473</td>
</tr>
<tr>
<td>simple</td>
<td>1s</td>
<td>50.1</td>
<td>3674</td>
</tr>
<tr>
<td>firstorder’</td>
<td>10s</td>
<td>69.6</td>
<td>5103</td>
</tr>
<tr>
<td>jprover</td>
<td>10s</td>
<td>56.1</td>
<td>4114</td>
</tr>
<tr>
<td>any</td>
<td></td>
<td>90.1</td>
<td>6609</td>
</tr>
</tbody>
</table>

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

proofs of lemmas in the Coq standard library. In our setting, the Ben-Yelles-type algorithm mentioned in the previous section tends to perform significantly better than the available Coq’s tactics. The results of the evaluation are presented in Table 2. Our tactic (yreconstr) manages to reconstruct about 88% of the reproved theorems. However, it needs to be remarked that if we use the advice obtained from ATP runs then about 50% of the the reproved theorems follow by a combination of hypothesis simplification, the tactics intuition, auto, easy, congruence and a few heuristics (tactic simple). Moreover, the yreconstr tactic without any hints (yreconstr0), i.e., without using any of the information obtained from ATP runs, achieves a success rate of about 26%. The reconstruction success rate of the firstorder tactic combined with various heuristics is about 70% if generic axioms for equality are added to the context (tactic firstorder’). The jp tactic (which integrates the intuitionistic first-order automated theorem prover JProver into Coq) combined with various heuristics and equality axioms (tactic jprover) achieves a reconstruction success rate of about 56%. This low success rate is explained by the fact that in contrast to the firstorder tactic the jp tactic cannot be parameterised by a tactic used at the leaves of the search tree when no logical rule applies.

Acknowledgments. We thank the organizers of the First Coq Coding Sprint, especially Yves Bertot, for the help with implementing Coq export plugins. We wish to thank Thibault Gauthier for the first version of the Coq exported data, as well as Claudio Sacerdoti-Coen for improvements to the exported data and fruitful discussions on Coq proof reconstruction. This work has been supported by the Austrian Science Fund (FWF) grant P26201.

References


