CoqHammer

Strong Automation for Program Verification in Coq

Łukasz Czajka
University of Copenhagen
Denmark
luta@di.ku.dk

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

Abstract

We present CoqHammer: the first full hammer system for the Coq proof assistant. The system translates Coq logic to untyped first-order logic and uses external automated theorem provers (ATPs) to prove the translations of user given conjectures. Based on the output of the ATPs, the conjecture is then re-proved in the logic of Coq using an eauto-type proof search algorithm. Together with machine-learning based selection of relevant premises this constitutes a full hammer system.

The performance of the overall procedure has been evaluated in a bootstrapping scenario emulating the development of the Coq standard library. Over 40% of the theorems in the Coq standard library can be proved in a push-button mode in about 40 seconds of real time on a 8-CPU system. This offers a huge saving of human work in programming language formalizations.

CCS Concepts • **Theory of computation → Type theory**; **Automated reasoning**;

Keywords hammer, automation, dependent type theory, translation, automated theorem prover

ACM Reference Format:
Łukasz Czajka and Cezary Kaliszyk. 2018. CoqHammer: Strong Automation for Program Verification in Coq. In *Proceedings of The Fourth International Workshop on Coq for Programming Languages (CoqPL'18).* ACM, New York, NY, USA, 2 pages. https://doi.org/

1 Introduction

Hammers provide most powerful general purpose automation for proof assistants based on HOL and set theory today [3]. A hammer system combines learning from previous proofs with translation of the problems to the logics of automated systems and reconstruction of the successfully found proofs. For many higher-order logic developments a third of the proofs can be proved by a hammer in push-button mode [2, 5]. Despite the gaining popularity of the more advanced versions of type theory, such as those based on the Calculus of Inductive Constructions, the construction of hammers for such foundations has been hindered so far by the lack of translation and reconstruction components. In this paper we present CoqHammer [4]: a full hammer system for the Coq proof assistant.

CoqPL'18, January 13, 2018, Los Angeles, CA, USA
2018.

2 Architecture

The goal of a hammer is for a context Γ which consists of all declarations accessible at a given point from the Coq kernel (typically there are thousands or tens of thousands of them), and a conjecture τ of type Prop, to find a term M such that $\Gamma \vdash M : \tau$. Hammers work in three phases.

Premise selection which heuristically chooses a subset of the accessible declarations that are likely useful for the conjecture τ. These declarations, together with the declarations they depend on, form a context $\Gamma_0 \subseteq \Gamma$. Typically, the size of Γ_0 is on the order of hundreds of declarations. In CoqHammer this phase is performed using machine learning algorithms: k-nearest neighbours or Sparse Naive Bayes.

Translation of the conjecture τ together with the context Γ_0 to the input formats of first-order automated theorem provers (ATPs) like Vampire [6] or Eprover [7], and running the ATPs on the translations. The reason for employing first-order ATPs is that they are currently the strongest and most optimised general-purpose automated theorem provers.

For CoqHammer we had to devise a new translation efficiently handling a reasonable subset of dependent type theory. The translation is neither sound nor complete, but it is sound and complete "enough" to be practically usable by a hammer tool. The ATPs employed are classical while the logic of Coq is intuitionistic, but this generates fewer problems than might be expected.

Reconstruction which uses the information obtained from a successful ATP run to re-prove the conjecture in the logic of the proof assistant or to directly reconstruct the proof term.

In CoqHammer proof reconstruction is performed by an auto-type algorithm implemented in Ltac. The proof search procedure does mostly backward Prolog-style reasoning – modifying the goal by applying hypotheses from the context. The core of our search procedure may be seen as an extension of the Ben-Yelles algorithm to first-order intuitionistic logic with all connectives [8]. Our implementation extends this core idea with various heuristics. We augment the proof search procedure with the use of existential metavariables like in `eauto`, a looping check, some limited forward

reasoning, the use of the congruence tactic, and heuristic rewriting using equational hypotheses. Our reconstruction tactics are also available for separate use, without invoking external ATPs.

The proof reconstruction phase does not assume any additional axioms. We re-prove the theorems in the intuitionistic logic of Coq, effectively using the output of the ATPs merely as hints. Currently, the only information from ATP runs we use is a list of lemmas needed by the ATP to prove the theorem (these are added to the context) and a list of constant definitions used in the ATP proof (we try unfolding these constants and no others).

This strategy is surprisingly successful, with over 85% of the theorems proven by the ATPs reconstructed in the logic of Coq.

3 Usage

A typical use of a hammer is to prove relatively simple goals using available lemmas. The problem is to find appropriate lemmas in a large collection of all accessible lemmas and combine them to prove the goal. An example of a goal solvable by our hammer, but not solvable by any standard Coq tactics, is the following.

```
forall (A : Type) (l1 l2 : list A)
       (x y1 y2 y3 : A),
  In x l1 \/ In x l2 \/ x = y1 \/
  In x (y2 :: y3 :: nil) ->
  In x (y1 :: (l1 ++ (y2 :: (l2 ++
                             (y3 :: nil)))))
```

The statement asserts that if x occurs in one of the lists l1, l2, or is equal to y1, or occurs in the list y2 :: y3 :: nil consisting of the elements y2, y3, then it occurs in the list

```
y1 :: (l1 ++ (y2 :: (l2 ++ (y3 :: nil))))
```

where ++ denotes list append and :: denotes the list cons operator. Eprover quickly finds a proof of the goal using six lemmas from the module Lists.List in the Coq standard library: in_nil, in_inv, in_cons, app_comm_cons, in_eq, in_or_app. The found ATP proof may be automatically reconstructed in Coq by our proof search procedure.

The advantage of a hammer is that it is a general system not depending on any domain-specific knowledge. The hammer plugin may use all currently accessible lemmas, including those proven earlier in a given formalization, not only the lemmas from the standard library or other libraries.

The CoqHammer tool is implemented as a Coq plugin. It provides a tactic hammer which invokes the external ATPs on a translation of the current goal, with the translations of the preselected lemmas as the axioms. It invokes a number of ATPs in parallel, and when one of them succeeds tries to reconstruct the proof in Coq using the returned dependencies. The reconstruction phase tries different variants of our proof search procedure with time limits for each reconstruction tactic. When one of the tactics succeeds it is displayed by CoqHammer in the response window. The user is then supposed to replace the hammer tactic with the displayed reconstruction tactic. The reconstruction tactic (usable also as a standalone tool) does not use any time limits or external ATPs – its success is reproducible on different machines

The source code of the CoqHammer plugin for Coq versions 8.5 and 8.6 is available at:

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

The hammer is able to re-prove completely automatically 40.8% of the Coq standard library proofs on a 8-CPU system in about 40 seconds. However, since we tested the plugin only on the Coq standard library during development, the success rate may be noticeably lower for other libraries, especially when dependent types are heavily used.

4 Conclusion

We presented CoqHammer: the first full hammer system for the Coq proof assistant. CoqHammer is a general-purpose automated reasoning system capable of re-proving over 40% of Coq standard library lemmas. This offers a substantial saving of human work, also in the programming language formalization effort. It also complements more domain specific automation, such as SMTCoq [1].

Acknowledgments

This work has been supported by the European Research Council (ERC) grant no. 714034 *SMART*. The first author was supported by Marie Skłodowska-Curie action "InfTy", program H2020-MSCA-IF-2015, number 704111.

References

[1] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. 2011. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In *Certified Programs and Proofs (CPP 2011) (LNCS)*, Jean-Pierre Jouannaud and Zhong Shao (Eds.), Vol. 7086. Springer, 135–150.

[2] Jasmin C. Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. 2016. A Learning-Based Fact Selector for Isabelle/HOL. *J. Autom. Reasoning* 57, 3 (2016), 219–244. https://doi.org/10.1007/s10817-016-9362-8

[3] Jasmin C. Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. 2016. Hammering towards QED. *J. Formalized Reasoning* 9, 1 (2016), 101–148. http://jfr.unibo.it/article/view/4593

[4] Łukasz. Czajka and Cezary Kaliszyk. 2017. Hammer for Coq: Automation for Dependent Type Theory. (2017). Submitted. Available at http://cl-informatik.uibk.ac.at/cek/coqhammer.

[5] Cezary Kaliszyk and Josef Urban. 2014. Learning-Assisted Automated Reasoning with Flyspeck. *J. Autom. Reasoning* 53, 2 (2014), 173–213. https://doi.org/10.1007/s10817-014-9303-3

[6] Laura Kovács and Andrei Voronkov. 2013. First-Order Theorem Proving and Vampire. In *CAV 2013 (LNCS)*, Natasha Sharygina and Helmut Veith (Eds.), Vol. 8044. Springer, 1–35.

[7] Stephan Schulz. 2013. System Description: E 1.8. In *LPAR 2013 (LNCS)*, Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov (Eds.), Vol. 8312. Springer, 735–743. https://doi.org/10.1007/978-3-642-45221-5_49

[8] Paweł Urzyczyn. 2016. Intuitionistic Games: Determinacy, Completeness, and Normalization. *Studia Logica* 104, 5 (2016), 957–1001.