ComplCoq: Rewrite Hint Construction with Completion Procedures

Mirai Ikebuchi
Massachusetts Institute of Technology
Cambridge, MA, USA
ikebuchi@mit.edu

Keisuke Nakano
Tohoku University
Sendai, Miyagi, Japan
k.nakano@acm.org

We present ComplCoq, a prototype Coq plugin for automated algebraic reasoning by completion procedures on rewriting systems. Completion procedures, originated by Knuth and Bendix [1], are ways to construct a terminating and confluent rewriting system which is equivalent to the input rewriting system. Terminating and confluent rewriting systems are called complete rewriting systems. ComplCoq provides vernacular commands for completion procedures on Coq terms whose types are in the form

```
forall x1 x2 ... xn, l = r
```

where `l` and `r` are first-order terms with variables `x1`, `x2`, ..., `xn`. We call such Coq terms rewrite rules. The vernacular commands automatically generate rewrite rules which form a complete rewriting system and add them into a hint rewrite database. So, users can use the hint database with `autorewrite` tactic and it automates proofs of algebraic equalities. Since `autorewrite` essentially normalizes goals with the rewriting system given as a hint database, the rewriting system should be terminating. In addition, if the rewriting system is not confluent, normal forms are not unique. Or equivalently, `autorewrite` does not generally reduce a goal `t = s` into a trivial form `t' = t'` even if `t = s` can be proved by the rewrite rules in the system. This is the reason why completion procedures are important for automation of algebraic reasonings.

For example, let `A` be a term of type `Set` and let `op : A -> A -> A` be a function that satisfies

```
ax : forall a b c, op (op a b) (op b c) = b.
```

Users call the completion by

```
Require Import Setoid Completion.Completion.
Complete ax : dbname sigs op.
```

Then, ComplCoq generates a complete rewriting system and registers it as a hint database `dbname`. If the users invoke

```
Print Rewrite HintDb dbname.
```

then Coq will return

```
Database dbname
rewrite -> ax of type forall a b c : A, op (op a b) (op b c) = b
rewrite -> dbname_lemma of type forall a a0 a1 : A,
op a0 (op (op a0 a) a1) = op a0 a
rewrite -> dbname_lemma0 of type forall a a0 a1 : A,
op (op a1 (op a0 a)) a = op a0 a
```

More generally, the command

```
Complete ax1 ax2 ... axn : dbname sigs f1 f2 ... fm
```

performs Knuth-Bendix completion on rewriting rules `ax1`, `ax2`, ..., `axn` with lexicographic path ordering with `f1 < f2 < ... < fn`. Since Knuth-Bendix completion does not always succeed, especially for non-orientable rewrite rules such as commutativity `f x y = f y x`, ComplCoq provides commands for more powerful completion procedures. The tactic `ordered_autorewrite` normalizes goals by ordered rewriting which terminates even

with non-orientable rewrite rules, and the command `OComplete` performs ordered completion (a.k.a. unfailing Knuth-Bendix completion) [2], a completion procedure based on ordered rewriting. For example, `OComplete` can produce complete rewriting systems for theories with commutativity such as semilattices and commutative groups, and `ordered_autorewrite` reduces goals into their unique normal forms. From this, a way to think of ComplCoq is that it gives a generalization of `ring_simplify`. Also, ComplCoq provides an experimental completion for associative and commutative theories which is more efficient than ordered completion on several basic theories like commutative groups and rings.

ComplCoq is available at `https://mir-ikbch.github.io/compl_coq/` and currently supports Coq v8.5 and v8.6.

References

[1] D. Knuth, P. Bendix. *Simple Word Problems in Universal Algebras*. In J. 828 Leech, editor, Computational Algebra, pages 263–297, 1970, Springer.

[2] L. Bachmair, N. Dershowitz, and D. A. Plaisted, *Completion without failure*. Resolution of Equation in Algebraic Structure, volumes 2, pages 1–30, 1989, Academic Press.