ComplCoq: Rewrite Hint Construction with Completion Procedures

Mirai Ikebuchi (MIT)
Keisuke Nakano (Tohoku University)
ComplCoq: A prototype Coq plugin that provides

1. vernacular commands for completion procedures
2. new tactics for ordered rewriting

Make your algebraic proofs shorter and simpler

(Currently supports v8.5 and v8.6)
Proof by Rewriting

Example 1 (Monoid)

\( S : \text{Set}, \ (\cdot) : S \rightarrow S \rightarrow S, \ e : S \)

\[ \text{assoc} : \forall x \ y \ z, \ (x \cdot y) \cdot z = x \cdot (y \cdot z) \]

\[ \text{id1} : \forall x, \ x \cdot e = x \]

\[ \text{id2} : \forall x, \ e \cdot x = x \]

------------------------------------------------

\[(x \cdot e) \cdot ((y \cdot z) \cdot (e \cdot w)) = x \cdot (y \cdot (z \cdot w))\]

How to prove?
(x * e) * ((y * z) * (e * w)) = x * (y * (z * w))
> rewrite id1.
x * ((y * z) * (e * w)) = x * (y * (z * w))
> rewrite id2.
x * ((y * z) * w) = x * (y * (z * w))
> rewrite assoc.
x * (y * (z * w)) = x * (y * (z * w))
> reflexivity.

But, we don’t usually give this kind of full proof steps in pencil-paper proofs.

This can be automated by

Hint Rewrite assoc id1 id2 : monoid.
autorewrite with monoid. reflexivity.
Hint Rewrite $\text{rule}_1 \ldots \text{rule}_n : \text{base}$
adds terms $\text{rule}_1, \ldots, \text{rule}_n$ in the rewrite hint DB $\text{base}$ with the left-to-right orientation.

Hint Rewrite $\leftarrow \text{rule}_1 \ldots \text{rule}_n : \text{base}$
same, but right-to-left

autorewrite with $\text{base}$
normalizes the goal with the rules in $\text{base}$
Example2 (Group)

\[ S : \text{Set}, \quad (\ast) : S \rightarrow S \rightarrow S, \quad e : S, \quad i : S \rightarrow S \]

assoc, id1, id2

\[ \text{inv} : \forall x, \ x \ast (i \ x) = e \]

\[ \begin{array}{c}
\text{(x \ast ((i \ x) \ast (y \ast z)))} \ast (i \ z) = y
\end{array} \]

Hint Rewrite assoc id1 id2 inv : group.

autorewrite with group.

\[ \begin{array}{c}
\text{---\rightarrow x} \ast ((i \ x) \ast y) = y
\end{array} \]

We need to use

assoc : \forall x y z, \ (x \ast y) \ast z = x \ast (y \ast z)

in right-to-left orientation.
Example2 (Group)

S : Set,  (*) : S -> S -> S,  e : S,  i : S -> S

assoc, id1, id2

inv : forall x, x * (i x) = e

----------------------------------------------
(x * ((i x) * (y * z))) * (i z) = y

Hint Rewrite assoc id1 id2 inv : group.
autorewrite with group.

--- > x * ((i x) * y) = y

We need to use

assoc : forall x y z, (x * y) * z = x * (y * z)

in right-to-left orientation.
Example2 (Group)

S : Set,  (•) : S -> S -> S,  e : S,  i : S -> S

assoc, id1, id2

inv : forall x, x • (i x) = e

---------------------------------------------
(x • ((i x) • (y • z))) • (i z) = y

Hint Rewrite id1 id2 inv : group.

Hint Rewrite <- assoc : group.

autorewrite with group.

----> (y • z) • (i z) = y
A simple solution (for this case)

Add a new rule

\[ \text{helper} : \forall x y, \; x \ast ((i \; x) \ast y) = y \]

into group.

\[ \text{goal:} \; (x \ast ((i \; x) \ast (y \ast z))) \ast (i \; z) = y \]

Hint Rewrite assoc id1 id2 inv helper : group.
autorewrite with group.

\[ \longrightarrow y = y \]

However, this does not solve \((i \; e) \ast x = x\) even though
\((i \; e) \ast x = e \ast ((i \; e) \ast x) = x\).
Completeness

We want our hint DB to have the property that for any two terms $t, s$, if $t = s$ can be proved by the rewrite rules in the hint DB, then $t$ and $s$ have the same normal form.

This is equivalent to termination + confluence. A terminating and confluent term rewrite system (TRS) is called a complete TRS.

We want a complete system for the group axioms.
The hint DB groupc which consists of

\[(x * y) * z = x * (y * z), \quad i e = e,\]
\[e * x = x, \quad x * ((i x) * y) = y,\]
\[x * e = x, \quad (i x) * (x * y) = y,\]
\[i (x * y) = (i x) * (i y), \quad x * (i x) = e,\]
\[i (i x) = x, \quad (i x) * x = e\]

is complete and equivalent to group.
That is, for every \(t_1, t_2\) which are equal to each other under group, \(t_1 = t_2\) can be proved by autorewrite with groupc. reflexivity.

How can we construct a complete TRS of a given TRS?
Ans. completion procedures
E.g. Group $R = \{\text{associativity, identity, inverse}\}$

\[(x \ast (i \ x)) \ast y\]

- **associativity**
  
  \[x \ast ((i \ x) \ast y)\]

- **inverse**
  
  \[y\]

: critical pair
E.g. Groups

\[(x \ast (i \ x)) \ast y\]

associativity

\[x \ast ((i \ x) \ast y)\]

inverse

\[y\]

new rule

: critical pair

Add \(x \ast ((i \ x) \ast y) = y\) into \(R\)
e * ((i e) * x)

identity

(i e) * x

added in the prev step

new critical pair

x

14 / 23
\(e \ast ((i \ast e) \ast x)\)

- identity
- added in the prev step

\((i \ast e) \ast x\) \rightarrow x

Add

... Repeat this until we have no critical pair \((t_1, t_2)\) with \(\hat{t}_1 \neq \hat{t}_2\)
where \(\hat{t}\) is the normal form of \(t\) wrt the current \(R\)
Then we will get

a complete TRS for the group axioms.
Orientation is important:

\[ id1 : \forall x, x \cdot e = x \]

Hint Rewrite <- id1 : base.
autorewrite with base.

\[ \Rightarrow x \rightarrow x \cdot e \rightarrow x \cdot (x \cdot e) \rightarrow \ldots \]

Fix a \textit{good} order \(>\) and use the orientation \(t \rightarrow s\) so that \(t > s\).

“Good” means well-founded, closed under context and substitution (reduction order)

Currently ComplCoq supports lexicographic path ordering (LPO)
Knuth-Bendix Completion (Sketch)

Input: \( R : \text{TRS}, > : \text{reduction order} \)
Output: A complete TRS \( R' \) equivalent to \( R \)

1. Find a critical pair \( t_1 \xleftarrow{l_1 \rightarrow r_1} t \xrightarrow{l_2 \rightarrow r_2} \)
2. Compute the normal forms \( \hat{t}_1, \hat{t}_2 \) of \( t_1, t_2 \) wrt \( R \).
3. Add \( \hat{t}_1 \rightarrow \hat{t}_2 \) or \( \hat{t}_2 \rightarrow \hat{t}_1 \) depending on \( > \)

Repeat until \( R \) does not have critical pairs with \( \hat{t}_1 \neq \hat{t}_2 \).
KB completion cannot handle non-orientable rules such as commutativity \( f \ x \ y = f \ y \ x \) (\( x, y \) are variables)

\[
f \ a \ b \rightarrow f \ b \ a \rightarrow f \ a \ b \rightarrow f \ b \ a \rightarrow \ldots
\]

Ordered Rewriting: \( t \) is ordered-rewritten to \( s \) by TRS \( R \) if \( t \) is (non-ordered-)rewritten to \( s \) by \( R \) and \( t > s \).

With LPO, if \( a, b \) are constants with \( a > b \),

\[
f \ a \ b \rightarrow f \ b \ a, \text{ but not } f \ b \ a \rightarrow f \ a \ b.
\]

Ordered Completion (Unfailing KB): KB completion using ordered rewriting
ComplCoq’s Vernacular Commands

Complete \( \text{rule}_1 \ldots \text{rule}_n : \text{base} \ \text{sig} \ c_1 \ldots c_m \)

OComplete \( \text{rule}_1 \ldots \text{rule}_n : \text{base} \ \text{sig} \ c_1 \ldots c_m \)

KB/ordered completion on rewrite rules \( \text{rule}_1 \ldots \text{rule}_n \) with LPO with \( c_1 < c_2 < \cdots < c_m \) (c’s are constants)
Automatically generate new rewrite rules along with their proofs

\( \text{rule’s} \) are terms of type \( \forall x_1 \ x_2 \ldots, \equiv t_1 \ t_2, \)
t1, t2 : terms with variables x1, x2, . . .,
\( \equiv \text{equiv} : \text{eq or symmetric setoid equality} \)
The result is added into base.
Implementation

- Allows only 1st order terms, i.e., terms consisting of only variables, constants and applications & variables don’t take arguments.
- Internally, variables are represented by evars.
- Unification: Use Coq’s internal unification
- Generates new rewrite rules with their proofs
  - Coq has API Pfedit.build_by_tactic which constructs a proof term from a tactic. → Call congruence
Tactics

- ordered_rewrite \( rule \) \( \text{sig} \) \( c_1 \ c_2 \ldots \ c_m \)
  ordered rewriting by \( rule \) with \( \text{LPO} \) with \( c_1 < c_2 < \cdots < c_m \)

- ordered_autorewrite \( \text{base} \) \( \text{sig} \) \( c_1 \ c_2 \ldots \ c_m \)
  ordered rewriting version of autorewrite

Implementation: Check the order between the goal and the new goal, call API for rewrite tactic if current goal \( > \) new goal.
Future Work

- Support for v8.7 and v8.8
- Use external completion tools such as mkbTT (Winkler, Sato, Middeldorp, Kurihara), MaxComp (Klein, Hirokawa), Slothrop (Wehrman, Stump, Westbrook), etc.
- Or, write & prove completion in Coq and do \textit{proof by reflection}
- Extending to non-first order terms, e.g.,
  \begin{verbatim}
  forall (A:Type)(x1 x2... : A),
  @eq A (f A x1 x2...) (f ...)
  \end{verbatim}