ComplCoq: Rewrite Hint Construction with Completion Procedures

Mirai Ikebuchi (MIT)
Keisuke Nakano (Tohoku University)

This Work

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

Example1 (Monoid)

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


assoc : forall x y z, (x * y) * z = x * (y * z)
id1 : forall x, x * e = x
id2 : forall x, e * x = x
-----------------------------------------------
(x * e) * ((y * z) * (e * w)) = x * (y * (z * 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.
```

Rewrite Hints and autorewrite

- `Hint Rewrite` $rule_1 \dots rule_n$: *base*

  adds terms $\text{rule}_1, \dots, \text{rule}_n$ in the rewrite hint DB *base* with the left-to-right orientation.

- `Hint Rewrite <-` $rule_1 \dots rule_n$: *base*

  same, but right-to-left

- `autorewrite with` *base*

  normalizes the goal with the rules in *base*

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 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

```
helper : forall x y, x * ((i x) * y) = y
```

into group.

```
goal: (x * ((i x) * (y * z))) * (i z) = y

Hint Rewrite assoc id1 id2 inv helper : group.
autorewrite with group.
---> y = y
```

However, this does not solve `(i e) * x = x` even though `(i e) * x = e * ((i e) * 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

$$
\begin{array}{rr}
\texttt{(x * y) * z = x * (y * z),} & \texttt{i e = e,} \\
\texttt{e * x = x,} & \texttt{x * ((i x) * y) = y,} \\
\texttt{x * e = x,} & \texttt{(i x) * (x * y) = y,} \\
\texttt{i (x * y) = (i x) * (i y),} & \texttt{x * (i x) = e,} \\
\texttt{i (i x) = x,} & \texttt{(i x) * x = e}
\end{array}
$$

is complete and equivalent to `group`.

That is, for **every** `t1, t2` which are equal to each other under `group`, `t1 = t2` 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 $R = \{\text{associativity, identity, inverse}\}$

$$\begin{tikzcd} & \texttt{(x * (i x)) * y} \arrow[dl, "\text{associativity}"'] \arrow[dr, "\text{inverse}"] & \\ \texttt{x * ((i x) * y)} & & \texttt{y} \end{tikzcd}$$

: critical pair

E.g. Groups

$$\begin{tikzcd}
& \texttt{(x * (i x)) * y} \arrow[dl, "\text{associativity}"'] \arrow[dr, "\text{inverse}"] & \\
\texttt{x * ((i x) * y)} \arrow[rr, red, "\text{new rule}"'] & & \texttt{y}
\end{tikzcd}$$

: critical pair

Add `x * ((i x) * y) = y` into R

$$\begin{tikzcd} & \texttt{e * ((i e) * x)} \arrow[dl, "\text{identity}"'] \arrow[dr, "\text{added in the prev step}"] & \\ \texttt{(i e) * x} & & \texttt{x} \end{tikzcd}$$

new critical pair

$$
\begin{tikzcd}
 & \texttt{e * ((i e) * x)} \arrow[dl, "\text{identity}"'] \arrow[dr, "\text{added in the prev step}"] & \\
\texttt{(i e) * x} \arrow[rr, "\text{Add}"'] & & \texttt{x}
\end{tikzcd}
$$

... 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 of Rewrite Rules

Orientation is important:

```
id1 : forall x, x * e = x

Hint Rewrite <- id1 : base.
autorewrite with base.
===>   x -> x * e -> x * (x * e) -> ...
```

Fix a *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 : TRS, $>$: reduction order

Output: A complete TRS R' equivalent to R

1. Find a critical pair $t_1 \xleftarrow{l_1 \to r_1} t \xrightarrow{l_2 \to r_2}$
2. Compute the normal forms $\hat{t_1}, \hat{t_2}$ of t_1, t_2 wrt R.
3. Add $\hat{t_1} \to \hat{t_2}$ or $\hat{t_2} \to \hat{t_1}$ 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 -> f b a -> f a b -> f b a -> ...
```

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 -> f b a`, but not `f b a -> f a b`.

Ordered Completion (Unfailing KB): KB completion using ordered rewriting

ComplCoq's Vernacular Commands

`Complete` $rule_1$ $\dots rule_n$: $base$ `sig` $c_1 \dots c_m$

`OComplete` $rule_1$ $\dots rule_n$: $base$ `sig` $c_1 \dots c_m$

KB/ordered completion on rewrite rules $rule_1 \dots rule_n$ with LPO with $c_1 < c_2 < \cdots < c_m$ (c's are constants)
Automatically generate new rewrite rules **along with their proofs**

rule's are terms of type `forall x1 x2 ..., equiv t1 t2`,
`t1, t2` : terms with variables `x1, x2, ...,`
`equiv` : `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* `sig` $c_1\ c_2 \ldots c_m$
  ordered rewriting by *rule* with LPO with $c_1 < c_2 < \cdots < c_m$
- `ordered_autorewrite` *base* `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.

Demo

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 *proof by reflection*
- Extending to non-first order terms, e.g.,
  `forall (A:Type)(x1 x2... : A),`
  `@eq A (f A x1 x2...) (f ...)`