Advanced tactics

Enrico Tassi
13 March
Outline

Bookkeeping
  Loading the goal
  Loading the context
  Idioms

Rewriting
  Matching
  Patterns
  Idioms
Terminology

The stack

\[ \begin{align*}
\text{Context} & \quad \begin{cases} 
    x : T \\
    S : \{\text{set } T\} \\
    xS : x \in S
\end{cases} \\
\text{Goal} & \quad \begin{cases} 
    \forall y, y == x \rightarrow y \in S
\end{cases}
\end{align*} \]

\textbf{The bar} \quad \text{-------------}

\textbf{Assumptions} \quad \textbf{Conclusion}

\textbf{Top} \quad \text{is the first assumption, } y \text{ here}

\textbf{Stack} \quad \text{alternative name for the list of Assumptions}
The real syntax of SSR

The real syntax is not this one:

\[
\begin{align*}
\text{move} & \Rightarrow x \ Hx \\
\text{move/andP} & : h \Rightarrow h \\
\text{case} & : x \\
\text{case/andP} & : x
\end{align*}
\]

These are compound tactics, the building blocks are:

- **move** and **case** are the tactics acting on Top
- : gen gen ... runs *before* the tactic to load the goal
- => ipat ipat ... runs *after* the tactic to load the context
- /andP is a view application on Top
Defective tactics

example

The implicit argument is Top:

case.

=================================

forall b : bool, P b
Defective tactics

example

The implicit argument is Top:

```
case.
```

\[
\begin{align*}
\forall b: \text{bool}, &\ P\ b \\
\text{P true} &\quad \text{P false}
\end{align*}
\]

Equivalent to:

```
move=>\Top.\ case: \Top.
```
Loading the goal
simple generalization

Slow motion for:

case: ab.

ab : A \( \land \) B

=============
G
Loading the goal
simple generalization

Slow motion for:

\textbf{case}: \texttt{ab}.

\[
\begin{align*}
\text{ab} & : A \land B \\
\text{----------} & \text{----------} \\
G & \quad A \land B \rightarrow G
\end{align*}
\]
Loading the goal

simple generalization

Slow motion for:

case: ab.

\[ ab : A \land B \]

\[ \begin{align*}
G \\
\begin{aligned}
A \land B \rightarrow G \\
A \rightarrow B \rightarrow G
\end{aligned}
\end{align*} \]
We can specify some items of the context that occur in the goal:

```
move: n m.
```

```
n : nat
m : nat
```

```
P n m
```
We can specify some items of the context that occur in the goal:

\texttt{move: n m.}

\begin{align*}
\text{n} & : \text{nat} \\
\text{m} & : \text{nat} \\
\text{P n m} & \quad \text{forall m, P n m}
\end{align*}
We can specify some items of the context that occur in the goal:

\textbf{move}: n \ m.

\begin{align*}
\text{n \ : \ nat} & \quad \text{m \ : \ nat} \quad \text{n \ : \ nat} \\
\text{---------} & \quad \text{----------------} \quad \text{---------------} \\
\text{P \ n \ m} & \quad \text{forall \ m, \ P \ n \ m} \quad \text{forall \ n \ m, \ P \ n \ m}
\end{align*}
We can specify the occurrences we want to grab, and to keep the context item:

move: n.+1 {1}m.

\[
\begin{align*}
n & : \text{nat} \\
m & : \text{nat} \\
=================
\end{align*}
\]

P n.+1 m m
We can specify the occurrences we want to grab, and to keep the context item:

\textbf{move: } n.+1 \{1\}m.

\begin{align*}
\text{n} & : \text{nat} & \quad \text{n} & : \text{nat} \\
\text{m} & : \text{nat} & \quad \text{m} & : \text{nat} \\
\text{forall } \text{m0}, \\
\quad \text{P n.+1 m m} & \quad \text{forall } \text{m0}, \\
\quad \text{P n.+1 m0 m} &
\end{align*}
We can specify the occurrences we want to grab, and to keep the context item:

\textbf{move: } n.+1 \{1\} m.

\begin{align*}
n & : \text{nat} & n & : \text{nat} & n & : \text{nat} \\
m & : \text{nat} & m & : \text{nat} & m & : \text{nat} \\
\text{P } n.+1 \text{ m m} & & \text{forall } m_0, & & \text{forall } n_0 m_0, \\
& & \text{P } n.+1 \text{ m0 m} & & \text{P } n_0 m_0 m
\end{align*}
We can generalize a lemma like
\( \text{ltnSn} : \text{forall} \ m, \ m < m.+1 \)

\text{move}: (\text{ltnSn} \ n).

\[ n : \text{nat} \]

\[ \text{P n} \]
We can generalize a lemma like
\( \text{ltnSn} : \text{forall } m, m < m.+1 \)

\textbf{move:} \((\text{ltnSn } n)\).

\[
\begin{align*}
\text{n : nat} & \quad \text{n : nat} \\
\text{======} & \quad \text{==========} \\
\text{P n} & \quad \text{n < n.+1 } \rightarrow \text{ P n}
\end{align*}
\]
Views
viewing Top differently

Views applied to Top:

case/andP.

a : nat
b : nat

==========
P a && P b -> G
Views

viewing Top differently

Views applied to Top:

case/andP.

\[
\begin{align*}
a & : \text{nat} & a & : \text{nat} \\
b & : \text{nat} & b & : \text{nat} \\
\text{========} & & \text{========} \\
P \ a \ \&\& \ P \ b \ \rightarrow \ G & & P \ a \ \&\ \& \ P \ b \ \rightarrow \ G
\end{align*}
\]
Views
viewing Top differently

Views applied to Top:

\[
\begin{align*}
& a : \text{nat} \\
& b : \text{nat} \\
& \text{=============} \\
& P \ a \ \& \ P \ b \ \rightarrow \ G
\end{align*}
\]
You have already seen that `elim/view` makes an exception:

```
elim/last_ind: s
```

What is an elimination principle?

```
last_ind : forall T (P : seq T -> Prop),
  P [::] ->
  (forall s x, P s -> P (rcons s x)) ->
  forall s : seq T, P s
```
The custom elimination principle can eliminate many items at the same time:

\[ \text{my\_ind} : \forall T P, \]
\[ P \text{[::]} \text{[::]} \rightarrow \]
\[ \left( \forall x \, xs \, y \, ys, \right. \]
\[ P \, xs \, ys \rightarrow P \left( x :: xs \right) \left( y :: ys \right) \rightarrow \]
\[ \forall s_1 \, s_2 : \text{seq} \, T, \, \text{size} \, s_2 = \text{size} \, s_1 \rightarrow P \, s_1 \, s_2 \]

\text{elim/my\_ind: s1 / s2.}
Loading the context views

Views can be applied in the middle of an intro pattern:

\[ \text{tactic} \Rightarrow a \ b \ \text{/andP} \ \text{pab} \ \text{qa} \]

\[ \forall a \ b : \text{nat}, \ P \ a \ \&\& \ P \ b \ \Rightarrow \ Q \ a \ \Rightarrow \ G \]

Equivalent to:

\[ \text{tactic} \Rightarrow a \ b. \ \text{move/andP} \Rightarrow \text{pab qa}. \]
Case analysis, usually to unpack, can be performed too:

\[\text{tactic} => \ a \ b \ /\text{andP}[pa \ pb] \ qa\]

\[
\begin{align*}
\forall a \ b : \text{nat}, \\
P\ a \ &\& \ P\ b \rightarrow \ Q\ a \rightarrow \ G
\end{align*}
\]

Equivalent to:

\[\text{tactic} => \ a \ b. \ \text{case}/\text{andP} => \ pa \ pb \ qa.\]
Loading the context

Real case analysis can be performed as follows:

\[
\text{tactic} \Rightarrow \ a \ [\text{Pa} \mid \text{Qa}]
\]

\[
\forall a : \text{nat}, \\
P a \lor Q a \rightarrow G
\]

Equivalent to:

\[
\text{tactic} \Rightarrow \ a. \ \text{case}.
\]
\[
\text{move} \Rightarrow \ Pa.
\]
\[
\ldots
\]
\[
\text{move} \Rightarrow \ Qa.
\]
\[
\ldots
\]
Loading the context

Case split (exception)

When the tactic is case or elim, brackets just after => do not perform (an additional) case analysis.

elim=> [ | x IH]
Loading the context
flags and combo

Cleanup flags:

//    gets rid of trivial goals
/=    simplifies the goals
//=    short for // and /=
{h}    clears h

Moreover: and => can be combined together:

elim: n => [ // | x IH] /=.
The goal can be prepared to obtain a stronger induction principle:

```
elim: n.+1 \{ -2 \} n \ (ltnSn n) \Rightarrow [\// \mid \{ n \} n \ IH j \ le_jn]
```

```
n : nat

------------
P n
```
The goal can be prepared to obtain a stronger induction principle:

\[ \text{elim: } n.+1 \{-2\} n \ (\text{lt}n Sn \ n) \Rightarrow [// | \{n\} n \ 	ext{IH} j \ le_jn] \]

\[
\begin{align*}
  n : \text{nat} \\
  \end{align*}
\]

\[
\begin{align*}
  n \ < \ n.+1 \rightarrow P \ n
\end{align*}
\]
The goal can be prepared to obtain a stronger induction principle:

```
elim: n.+1 \{\neg2\}n (lt\text{nSn n}) => [// | \{n\} n IH j le_jn]
```

```
n : nat
```

```
forall m, m < n.+1 -> P m
```
The goal can be prepared to obtain a stronger induction principle:

\[\text{elim: } n.\ +1 \ \{-2\}n \ (\text{ltnSn} \ n) \Rightarrow [// \ | \ \{n\} \ n \ \text{IH} \ j \ \text{le}_{-jn}]\]

\[n : \text{nat}\]

\[\text{forall } i \ m, \ m < i \rightarrow P \ m\]
The goal can be prepared to obtain a stronger induction principle:

\[
\text{elim: } n.+1 \{ -2 \} n \ (\text{lt} n \ S n \ n) \Rightarrow \begin{cases} &/\ / \mid \{ n \} \ n \ \text{IH} \ j \ \text{le}_j n \end{cases} \\
\end{cases}
\]

\[
\begin{array}{c}
n : \text{nat} \\
\text{forall } m, \\
\text{m < 0 } \rightarrow \ P \ m
\end{array}
\]

\[
\begin{array}{c}
n : \text{nat} \\
\text{forall } i, \\
\text{(forall } m, \ m < i \rightarrow P \ m) \rightarrow \\
\text{forall } m, \ m < i.+1 \rightarrow P \ m
\end{array}
\]
The goal can be prepared to obtain a stronger induction principle:

```
elim: n.+1 {-2}n (ltnSn n) => [// | {n} n IH j le_jn]
```

```
n : nat
IH : forall m, m < n -> P m
j : nat
le_jn : j < n.+1
```

```
P j
```
Equations can be substituted on the fly, and unneeded hypotheses cleared

```
case: ex => y [-> yA] {x}
```

\[\begin{align*}
x & : T \\
ex & : \exists y : T, \quad x = f \circ^{-1} y \land y \in A \\
& \quad f \circ x \in A
\end{align*}\]
Equations can be substituted on the fly, and unneeded hypotheses cleared

```plaintext
case: ex => y [-> yA] {x}

x : T
ex : exists y : T,
   x = f @*^-1 y /
   y \in A

f @* x \in A
```

```plaintext
y : T
yA : y \in A
```

```plaintext
f @* (f @*^-1 y) \in A
```
Idioms
Hypotheses refinement & substitution

The `have` tactic accepts the same flags of `=>`. The context can be refined and kept clean with `have`:

```
have {hyp1 hyp2} hyp3 : statement
  ...
  ...
```

Another example is with one shot equations.

```
have /andP[pa /eqP-> {b}] : P a && b == a
  ...
  ...
```
Outline

**Bookkeeping**
- Loading the goal
- Loading the context
- Idioms

**Rewriting**
- Matching
- Patterns
- Idioms
Ambiguity
Instantiation and occurrence

Lemma addnC \( x \ y : x + y = y + x \). Proof. ... Qed.
Lemma mulnC \( x \ y : x \times y = y \times x \). Proof. ... Qed.

Lemma ex a b : \((a + b)^2 = (c + d) \times (a + b)\).
Proof. rewrite addnC.
Ambiguity
Instantiation and occurrence

Lemma \text{addnC} \ x \ y : x + y = y + x. Proof. \ldots \ Qed.
Lemma \text{mulnC} \ x \ y : x \times y = y \times x. Proof. \ldots \ Qed.

Lemma \text{ex} \ a \ b : (a + b)^2 = (c + d) \times (a + b).
Proof. rewrite (\text{addnC} \ _ \ _).

The pattern \(_ + \_\) has many matches:
Ambiguity

Instantiation and occurrence

Lemma \texttt{addnC} \ x \ y : \ x + y = y + x. Proof. ... Qed.

Lemma \texttt{mulnC} \ x \ y : \ x * y = y * x. Proof. ... Qed.

Lemma \texttt{ex} \ a \ b : \ (a + b)^2 = (c + d) * (a + b).
Proof. rewrite \texttt{(addnC _ _)}.

The pattern \texttt{(_ + _)} has many matches:

\texttt{(a + b)^2 = (c + d) * (a + b)}
Ambiguity

Instantiation and occurrence

Lemma addnC x y : x + y = y + x. Proof. ... Qed.
Lemma mulnC x y : x * y = y * x. Proof. ... Qed.

Lemma ex a b : (a + b)^2 = (c + d) * (a + b).
Proof. rewrite (addnC _ _).

The pattern (_ + _) has many matches:
(a + b)^2 = (c + d) * (a + b)
(a + b)^2 = (c + d) * (a + b)
Ambiguity
Instantiation and occurrence

Lemma `addnC x y : x + y = y + x`. Proof. ... Qed.
Lemma `mulnC x y : x * y = y * x`. Proof. ... Qed.

Lemma `ex a b : (a + b)^2 = (c + d) * (a + b)`. Proof. `rewrite (addnC _ _)`.

The pattern `_ + _` has many matches:

(a + b)^2 = (c + d) * (a + b)
(a + b)^2 = (c + d) * (a + b)
(a + b)^2 = (c + d) * (a + b)
Lemma addnC x y : x + y = y + x. Proof. ... Qed.
Lemma mulnC x y : x * y = y * x. Proof. ... Qed.

Lemma ex a b : (a + b)^2 = (c + d) * (a + b).
Proof. rewrite (mulnC _ _).

The pattern (_, * _) has many matches:
Ambiguity
Instantiation and occurrence

Lemma addnC x y : x + y = y + x. Proof. ... Qed.
Lemma mulnC x y : x * y = y * x. Proof. ... Qed.

Lemma ex a b : (a + b)^2 = (c + d) * (a + b).
Proof. rewrite (mulnC _ _).

The pattern (_ * _) has many matches:
(a + b)^2 = (c + d) * (a + b)
Ambiguity
Instantiation and occurrence

Lemma \text{addnC} x y : x + y = y + x. Proof. ... Qed.
Lemma \text{mulnC} x y : x * y = y * x. Proof. ... Qed.

Lemma \text{ex} a b : (a + b)^2 = (c + d) * (a + b).
Proof. rewrite (\text{mulnC } _ _).

The pattern \(_ * _\) has many matches:
\((a + b)^2 = (c + d) * (a + b)\)
\((a + b)^2 = (c + d) * (a + b)\)
The SSR approach

It’s all about patterns:

- Inferred looking at the rewrite rule
- Eventually overridden by the user

Matching discipline:

1. Traverse the goal outside in, left to right
2. Look for verbatim copies of the key of the pattern
   e.g. \((_ + _)\)
3. There you match up to computation
4. If the matching fails, try the next occurrence of the key
5. If the matching succeeds, that subterm is the *only* instance of the pattern considered.

Note: the instance of the pattern may occur multiple times
Inferred pattern

Recall the goal:

\[(a + b)^2 = (c + d) + (a + b)\]

To rewrite there we can decorate the rule with a pattern:

\textit{rewrite addnC}

The first instance of the pattern \((_ + _)\) is:

\[(a + b)^2 = (c + d) + (a + b)\]

That occurs twice:

\[(a + b)^2 = (c + d) + (a + b)\]

The result is:

\[(b + a)^2 = (c + d) + (b + a)\]
Recall the goal and consider the target:

\[(a + b)^2 = (c + d) + (a + b)\]

To rewrite there we can decorate the rule with a pattern:

`rewrite [c + _] addnC`

The pattern `[c + _]` selects:

\[(a + b)^2 = (c + d) + (a + b)\]

The result is:

\[(a + b)^2 = (d + c) + (a + b)\]
Simple pattern
forcing unfolding

Recall the goal and our target:

**Lemma** \(\text{sumn\_nseq} \ x \ n : \text{sumn} (\text{nseq} \ n \ x) = x \times n\)

\((a + b)^2 = (c + d) + (a + b)\)

The *key* of the given pattern differs from the inferred one:

**rewrite** \[-[_^2]\text{sumn\_nseq}\]

The pattern \([-^2]\) selects:

\(_\_\_\_\_\_\_\) \((a + b)^2 = (c + d) + (a + b)\)

The result is:

\(\text{sumn} (\text{nseq} (a + b) (a + b)) = (c + d) + (a + b)\)
Simple contextual pattern

Assume you want to rewrite only the blue subterm:

\[(a + b)^2 = (c + d) + (a + b)\]

Instead of using the occurrence number \{1\} to identify the occurrence, one can use its context:

\texttt{rewrite [in \_\_\_^2]addnC}

The pattern selects:

\[(a + b)^2 = (c + d) + (a + b)\]

Then all its subterms are matched against the inferred pattern \(_ + _\):

\[(b + a)^2 = (c + d) + (a + b)\]
Precise contextual pattern

Assume you want to rewrite only the blue subterm:

Lemma addn0 n : n + 0 = n.

\[(a + b)^2 = (c + d) + (a + b)\]

We can identify the occurrence of \(b\) using its context:

rewrite \(-[X \ in \ (_ + X)^2]addn0\)

The pattern selects:

\[(a + \_\_\_\_\_)^2 = (c + d) + (a + b)\]

The \(X\) selects:

\[(a + b)^2 = (c + d) + (a + b)\]

Then the substitution happens exactly there:

\[(a + (b + 0))^2 = (c + d) + (a + b)\]
Extra flags

Rewrite rules can be interleaved with other flags:

- To unfold/fold (local) definitions
  
  `rewrite /def ~/def`

- To simplify (cleanup) the goal or get rid of trivial goals
  
  `rewrite /= //`

- To iterate 1 or more (or zero or more) times
  
  `rewrite !lem n?lem`

- To clear unneeded hypotheses
  
  `rewrite {h}`

Note that `/def` and `//=` can be decorated with patterns to restrict their action to a portion of the goal.
Idiom
Goal readability

Give short names to big expressions:

```coq
define d := gcd _ _.

n : nat
m : nat

define d := gcd n m

---------
gcd n m %| n
---------
d %| n
```

Unfold/fold when needed:

```coq
rewrite /d
rewrite -/d
```
Idiom

Rules with premises

Consider the goal:

\[ b_{gt0} : 0 < b \]

\[
(a + b) \times c \text{ } \%\text{ } / (a + b) = c + 0.
\]

And the lemmas:

\[ \text{mulKn } m \text{ } d : 0 < d -> (d \times m) \text{ } \%\text{ } / \text{ } d = m \]

\[ \text{ltn_addl } m \text{ } n \text{ } p : m < n -> m < p + n \]

We chain the two rules to kill the side condition:

\[ \text{rewrite } \text{mulKn } ?\text{ltn_addl } //. \]

The first rule leaves two active goals:

\[ c = c + 0 \]

\[ 0 < a + b \]
Idiom

Rules with premises

Consider the goal:

\[ b_{\text{gt0}} : 0 < b \]

\[
(a + b) \times c \mod (a + b) = c + 0.
\]

And the lemmas:

\[ \text{mulKn } m \ d : 0 < d \rightarrow (d \times m) \mod d = m \]
\[ \text{ltn_addl } m \ n \ p : m < n \rightarrow m < p + n \]

We chain the two rules to kill the side condition:

\text{rewrite mulKn} \ ?\text{ltn_addl} \ //.

The second (optional) rule leaves three active goals:

\[ c = c + 0 \]
\[ \text{true} \]
\[ 0 < b \]
Idiom

Rules with premises

Consider the goal:

\[ b_{\text{gt}0} : 0 < b \]

\[ (a + b) \times c \%/(a + b) = c + 0. \]

And the lemmas:

mulKn \( m \ d \) : \( 0 < d \rightarrow (d \times m) \%/ d = m \)

ltn_addl \( m \ n \ p \) : \( m < n \rightarrow m < p + n \)

We chain the two rules to kill the side condition:

\( \text{rewrite mulKn ?ltn_addl //}. \)

The cleanup switch \( // \) leaves only the main goal:

\[ c = c + 0 \]
Summary
What you should try to remember

move: \( (t) \Rightarrow \forall [t_1 \mid t_2] \).

elim/v: \{occ\}n n.+1 (ltnSn n) \Rightarrow [ \mid m \text{ IH }] //=.

rewrite lem ?lem !lem [pat]lem \[X \text{ in } pat\]lem.

Appetizers (for experts, see the manual):

rewrite (_ : a = b) [in X in pat]lem -[pat]/def
rewrite [_ a b]lem (lem1,lem2)
rewrite lem in hyp |- *
congr (_ && _)