The Importance of Being Formal

Martin Henz

July 4, 2013

# Cheat Sheet: Traditional Logic

The following explanation looks at each Coq tactic used so far and explains it in isolation. The aim of this is to help you in your Coq homework; it can be daunting to learn so many new commands in one go. Let me know if you have any questions or suggestions how to improve this document.

`unfold` *d*. Replace a term in the goal with its `definition`. Example: `unfold immortal.`
To do the replacement in a hypothesis *H* instead of the goal, use `unfold` *d* `in` *H*

`intro.` `intros.` Move a universal quantification (`forall`) or implication from beneath the bar to a hypothesis above the bar. When applied to `forall`, we are talking about the meta-level, as far as traditional logic is concerned. For example consider

```
Lemma SomeNon: forall subject object,
     Some non object are non subject
    -> Some non subject are not object.
```

You need to prove:

```
forall subject object,
     Some non object are non subject
    -> Some non subject are not object.
```

Applying `intro` twice will eliminate the declaration of the meta-variables `subject` and `object` above the bar, and replace them by the declaration of the variables `subject` and `object` above the bar, indicating that these two variables can be freely used in the rest of the proof. Now you are left with the proof obligation:

```
Some non object are non subject
-> Some non subject are not object.
```

Again, `intro` can be applied here. In order to prove that $P$ `->` $C$ (in English: "under the assumption $P$, we can prove $C$"), we of course need to *make* the assumption $P$, and then prove $C$. That means we move $P$ above the bar and are left with the proof obligation $C$. In our example, this looks like this:

```
Some non object are non subject
-------------------------------
Some non subject are not object
```

The overall proof looks like this:

```
Proof.
intro.
intro.
intro.
apply ObvE3.
apply ConvE1.
apply H.
Qed.
```

`intro` does its work only once while `intros` does it as many times as it can. We can therefore simplify our proof by writing:

```
Proof.
intros.
apply ObvE3.
apply ConvE1.
apply H.
Qed.
```

`apply` $X$**.** The simplest case is when the proof obligation is identical with a hypothesis, axiom or previously proven lemma $P$. In that case, the proof is simply `apply` $P$.

If we have a hypothesis `H : A -> B` (or axiom, or previously proven lemma), and our goal is `B`, then `apply H` will solve our goal and present us with a new goal, `A`. If we need to provide some additional hints as to how to use `H`, we can use `apply H with (...)` as in the following example:

```
Axiom Barbara : forall major minor middle,
    All middle are major
 /\ All minor are middle
 -> All minor are major.

Lemma GreeksMortality : All Greeks are mortal.

Proof.
apply Barbara with (middle := humans).
```

`rewrite` *X*`.` Used to substitute an equality into the goal. If we have `H : A = B` as a hypothesis (or axiom, previous lemma, etc.), and our goal contains `... A ...` then `rewrite H` will transform our goal into `... B ...`. For example, if the goal is

```
All humans are non non mortal.
```

then the tactic

```
rewrite NonNon.
```

will tranform the goal to

```
All humans are mortal.
```

`split.` Used to prove a conjunction, `/\`, or a double-implication `<->` (which actually is a conjunction of two implications), in the goal. The conjunction is split into two proof obligations, which then can be proven separately. For example, consider `Barbara`:

```
Axiom Barbara : forall major minor middle: Thing -> Prop,
    All middle are major
 /\ All minor are middle
 -> All minor are major.

Lemma GreeksMortality : All Greeks are mortal.
Proof.
apply Barbara with (middle := humans).
split.
```

After this, the proof obligation consists of

```
All humans are mortal /\ All Greeks are humans
```

In order to prove each part of this conjunction separately, we use the tactic

```
split.
```

The whole proof looks like this:

```
Proof.
apply Barbara with (middle := humans).
split.
apply HumansMortality.
apply GreeksHumanity.
Qed.
```

`destruct` *H*. Arguments may come with multiple premises. When you `intro` an argument, its premisses all end up in the same line as a hypothesis above the bar, separated by the symbol `/\`. In order to use each premise separately, you can apply `destruct` to the corresponding hypothesis. For example, let us say see the following premise on the right hand side in the Coq window:

```
H : No ducks are things_that_waltz /\
    No officers are non things_that_waltz /\ All my_poultry are ducks
```

We usually want to use each part separately. We can do that using:

```
destruct H.
```

After that, the premises look like this:

```
H : No ducks are things_that_waltz
H0 : No officers are non things_that_waltz /\ All my_poultry are ducks
```

Finally, we

```
destruct H0.
```

This allows us to refer to each of the premises separately:

```
H : No ducks are things_that_waltz
H0 : No officers are non things_that_waltz
H1 : All my_poultry are ducks
```

`admit.` Proves anything. Useful for instructors to give problem sets and when you want to prove goals in different orders than Coq gives you before coming back to solve the previous goals. Automatically get zero points for a homework problem when you include an `admit` in a solution.