Cheat Sheet Predicate Logic

Martin Henz, 10/7/2013

Bottom-up

General tactic on bottom-up reasoning: When you are proving a goal, and you think you need and can prove a hypothesis \( \phi \), you can add it via `assert \ \phi`. Coq will then ask you to first prove \( \phi \). After that, you can work on your original goal, now with the additional hypothesis of \( \phi \). This is a special case of \( \rightarrow \ e \), see below.

\( \top \): trivial.

\( \land \): split.

\( \land e_1 \): Let us call the goal \( g \). Here is how to use \( \land e_1 \) bottom-up, in case you would ever need it: `assert \ g \land \ \phi`. prove conjunction, `destruct H1. apply H2`. where \( H1 \) is the conjunction and \( H2 \) is \( g \).

\( \land e_2 \): Let us call the goal \( g \). Here is how to use \( \land e_2 \) bottom-up, in case you would ever need it: `assert \ \phi \land \ g`. prove conjunction, `destruct H1. apply H2`. where \( H1 \) is the conjunction and \( H2 \) is \( g \).

\( \lor \): left.

\( \lor \): right.

\( \lor e \): `destruct H`.

\( \rightarrow \): intro.

\( \rightarrow e \): `apply H`. (\( H \) is the implication)

\( \rightarrow e \): A variant of the rule allows you to prove a goal \( \psi \), by proving first \( \phi \), and then \( \phi \rightarrow \psi \): `assert \ \phi`, then prove \( \phi \), and finally prove goal \( \psi \) using \( \phi \)

\( \neg \land e \): `assert \ \phi \land \neg \phi`. `split`. prove \( \phi \) and \( \neg \phi \) separately, then use `destruct H1. contradiction H2`, where \( H1 \) is the asserted conjunction, and \( H2 \) is one part of it.

\( \neg i \): unfold not. intro.

\( \bot e \): exfalso.

\( \neg \neg e \): Let us call the goal \( g \). Here is how to use \( \neg \neg e \) bottom-up, in case you would ever need it: `assert (\neg \neg \ g)`. prove \( \neg \neg \ g \). Now use: `tauto`.

\( = i \): trivial.

\( = e \): `rewrite H`. (use `rewrite <- H` to apply equality \( H \) from right to left)

\( \forall e \): `apply H`.

\( \forall i \): intro.

\( \exists i \): `exists t`.

\( \exists e \): `destruct H`.

Derived rule: LEM + \( \forall e \): LEM (\( \phi \)).
Top-down

Coq allows you to apply some rules within the hypotheses, which makes many proofs a lot shorter. Here are some common uses of top-down reasoning:

→ e: `spec H1 H2`. (H1 is the implication)

¬i: `unfold not in H`.

∧i: `destruct H`.

= e: `rewrite H1 in H2`. (apply equation H1 in H2; use `rewrite <- H1 in H2` to apply equality H1 from right to left)

∀e: `spec H t`.