COQ CHEATSHEET

Jules Jacobs

December 9, 2021

CONTENTS

1 INTRODUCTION

This is Coq code that proves the strong induction principle for natural numbers:

```
From Coq Require Import Lia.

Lemma strong_induction (P : nat -> Prop) :
  (forall n, (forall m, m < n -> P m) -> P n) -> forall n, P n.
Proof.
  intros H n.
  eapply H.
  induction n.
  - lia.
  - intros m Hm.
    eapply H.
    intros k Hk.
    eapply IHn. lia.
Qed.
```

Coq proofs manipulate the *proof state* by executing a sequence of *tactics* such as `intros`, `eapply`, `induction`. Coq calculates the proof state for you after executing each tactic. Here's what Coq displays after executing the second `intros m Hm`.:

```
P: nat -> Prop
H: forall n : nat, (forall m : nat, m < n -> P m) -> P n
n: nat
IHn: forall m : nat, m < n -> P m
m: nat
Hm: m < S n
----------------------------
P m
```

The proof state consists of a list of variables and hypotheses above the line, and a goal below the line. A tactic may create 0, 1, 2, or more subgoals. A goal is solved if we succesfully apply a tactic that creates no subgoals (such as the `lia` tactic). Some tactics create multiple subgoals, such as the `induction` tactic: it creates one subgoal for the base case of the induction, and one subgoal for the inductive case.[1] We have to solve all the subgoals with a bulleted list of tactic scripts:

```
tac1.
+ tac2.
+ tac3.
+ tac4.
```

Bullets can nested by using different bullets for different levels (`-`, `+`, `*`):

```
tac1.
+ tac2.
  * tac3
  * tac4.
+ tac5.
```

We can also enter subgoals using brackets:

```
tac1.
{ tac2. }
{ tac3. }
tac4.
{ tac5. }
tac6.
```

This is most useful for solving side conditions. With bullets, we get a deep level of nesting if we have a sequence of tactics with side conditions. With brackets, we do not need to enclose the last subgoal in brackets, thus preventing deep nesting.

2 GOAL TACTICS

Goal	Tactic
$P \to Q$	`intros H`
$\neg P$	`intros H`
$\forall x, P(x)$	`intros x`
$\exists x, P(x)$	`exists x`, `eexists`
$P \land Q$	`split`
$P \lor Q$	`left`, `right`
Q	`apply H`, `eapply H` (where $H : (...) \to Q$ is a lemma or hypothesis with conclusion Q)
False	`apply H`, `eapply H` (where $H : (...) \to \neg P$ is a lemma or hypothesis with conclusion $\neg P$)
Any goal	`exfalso` (turns any goal into False)
Skip goal	`admit` (skips goal so that you can work on other subgoals)

1 Coq allows us to do induction not only on natural numbers, but also on other data types. Induction on other data types may create any number of subgoals, one for each constructor of the data type.

3 HYPOTHESIS TACTICS

Hypothesis	Tactic
$H : \mathit{False}$	`destruct H`
$H : \exists x, P(x)$	`destruct H as [x H]`
$H : P \wedge Q$	`destruct H as [H1 H2]`
$H : P \vee Q$	`destruct H as [H1|H2]`
$H : \forall x, P(x)$	`specialize (H y)`
$H : P \rightarrow Q$	`specialize (H G)` (where $G : P$ is a lemma or hypothesis)
$H : P$	`apply G in H`, `eapply G in H` (where $G : P \rightarrow (...)$ is a lemma or hypothesis)

4 EQUALITY, REWRITING, AND COMPUTATION RULES

Tactic	Meaning
`reflexivity`	Solve goal of the form $x = x$ or $P \leftrightarrow P$
`symmetry`	Turn goal $x = y$ into $y = x$ (or $P \leftrightarrow Q$)
`symmetry in H`	Turn hypothesis $H : x = y$ into $H : y = x$ (or $P \leftrightarrow Q$)
`unfold f`	Replace constant f with its definition (only in the goal)
`unfold f in H`	Replace constant f with its definition (in hypothesis H)
`unfold f in *`	Replace constant f with its definition (everywhere)
`simpl`	Rewrite with computation rules (in the goal)
`simpl in H`	Rewrite with computation rules (in hypothesis H)
`simpl in *`	Rewrite with computation rules (everywhere)
`rewrite H.`	Rewrite `H : x = y` (in the goal).
`rewrite H in G.`	Rewrite `H : x = y` (in hypothesis G).
`rewrite H in *.`	Rewrite `H1` (everywhere).
`rewrite <-H.`	Rewrite `H : x = y` backwards.
`rewrite H,G.`	Rewrite using `H` and then `G`.
`rewrite !H.`	Repeatedly rewrite using `H`.
`rewrite ?H.`	Try rewriting using `H`.
`subst`	Substitute away all equations `H : x = A` with a variable on one side.
`injection H as H`	Use injectivity of C to turn $H : C\ x = C\ y$ into $H : x = y$.
`discriminate H`	Solve goal with inconsistent assumption $H : C\ x = D\ y$.
`simplify_eq`	Automated tactic that does `subst`, `injection`, and `discriminate` automatically.

5 INDUCTIVE TYPES AND RELATIONS

Assume that Foo is some inductive type or $\mathrm{Foo}\ x\ y$ is an inductive relation.

Goal	Tactic
Foo x y	`constructor`, `econstructor` (tries to solve goal using some constructor of Foo)

Hypothesis	Tactic
H : Foo	`destruct H as [a b|c d e|f]`
H : Foo	`induction H as [a b IH|c d e IH1 IH2|f IH]`
H : Foo x y	`inversion H` (use when x,y are fixed terms)
H : Foo x y	`induction H` (use when x,y are variables)

It is often useful to define the tactic `Ltac inv H := inversion H; clear H; subst.` and use this instead of `inversion`.

The following tactics are useful to obtain the correct induction hypothesis:

Hypothesis	Tactic
H : P	`revert H` (opposite of `intros H`: turn goal Q into $P \to Q$)
x : A	`revert x` (opposite of `intros x`: turn goal Q into $\forall x, Q$)
H : P, x : A	`clear H`, `clear x` (remove hypothesis H or variable x)

6 INTRO PATTERNS

The `destruct x as pat` and `intros pat` tactics can unpack multiple levels at once using *intro patterns*. The intros tactic can be chained: `intros x y z.` ≡ `intros x. intros y. intros z.`

Data	Pattern
$\exists x, P$	`[x H]`
$P \land Q$	`[H1 H2]`
$P \lor Q$	`[H1|H2]`
$x = y$	`->` or `<-`
Inductive type	`[a b|c d e|f]`

Furthermore, `(x & y & z & ...)` is equivalent to `[x [y [z ...]]]`.

7 FORWARD REASONING

Tactic	Meaning
`assert P as H`	Create new hypothesis H : P after proving subgoal P
`cut P`	Split goal Q into two subgoals $P \to Q$ and P

Intro patterns can be used in combination with the `assert` tactic, e.g. `assert (A = B) as ->` or `assert (exists x, P) as [x H]`.

8 COMPOSING TACTICS

Tactic	Meaning
`tac1; tac2`	Do `tac2` on all subgoals created by `tac1`.
`tac1; [tac2|..]`	Do `tac2` only on the first subgoal.
`tac1; [..|tac2]`	Do `tac2` only on the last subgoal.
`tac1; [tac2|..|tac3|tac4]`	Do tactics on corresponding subgoals.
`tac1; [tac2|tac3..|tac4]`	Do tactics on corresponding subgoals.
`tac1 || tac2`	Try `tac1` and if it fails do `tac2`.
`try tac1`	Try `tac1`, and do nothing if it fails.
`repeat tac1`	Repeatedly do `tac1` until it fails.
`progress tac1`	Do `tac1` and fail if it does nothing.

9 AUTOMATION WITH `eauto`

The `eauto` tactic tries to solve goals using `eapply`, `reflexivity`, `eexists`, `split`, `left`, `right`. You can specify the search depth using `eauto n` (the default is $n = 5$).

You can give `eauto` additional lemmas to use with `eauto using lemma1,lemma2`. You can also use `eauto using foo` where `foo` is an inductive type. This will use all the constructors of `foo` as lemmas.

10 COMMON ERROR MESSAGES

Please submit your errors to me so that I can add them to this section.

You can also suggest additional content.

julesjacobs@gmail.com