Program Verification in Coq

The Coq Tactic Cheat Sheet

Propositional calculus

`intro`

`intro` shrinks the goal while introducing new information inside the context. It corresponds to the $\implies$-intro, $\neg$-intro, or $\forall$-intro rule of Natural Deduction.

When the current goal is of the form :

- `A -> B`, loads a hypothesis `A` in the context, and replaces the goal with `B`.
- `~A`, loads a hypothesis `A` into the context, and replaces the goal with `False`.
- `forall x : T, A x`, loads a new declaration of the form `x : T` into the context and replaces the goal with `A x`.

`intros`

The `intros` tactic repeats `intro` until it fails. It introduces zero or more items into the context from the constructs listed in intro. it never fails and may leave the context unchanged.

`exists`

`exists M` (where M is a term of type `T`) implements the $\exists$-intro rule of natural deduction.

When the goal is of the form `exists x : T, A x`, the tactic `exists M` replaces the current goal by a goal of the form `A M`.

`destruct`

The `destruct H` tactic (where `H` is the name of a hypothesis of the context) applies among the rules $\wedge$-left, $\implies$-left and $\exists$-left of Natural Deduction.

When `H` is of the form :

- `A /\B` replaces `H` with two new hypotheses `A` and `B`.
- `A \/B` replaces the current subgoal with two new subgoals, in which `H` is replaced by `A` and `B` respectively.
- `exists x : T, A x` introduces into the context a statement of the form `x: T` and replaces `H` with a new hypothesis of the form `A x`.

`split`

`split` replaces the current goal by two sub-goals :

When the current goal is of the form :

- `A /\B` creates a sub-goal for `A` and a sub-goal for `B`.
- `A <-> B` creates a sub-goal for `A -> B` and a sub-goal for `B -> A`.

`left` & `right`

When the goal is of the form `A \/ B`,

- the `left` tactic replaces the goal with the left side of the disjunction `A`
- the `right` tactic replaces the goal with the right side of the disjunction `B`

`apply`

`apply H` (where `H` is the name of a hypothesis of the context) applies one of the following rules :

If `H` is of the form :

- `A -> B` and if the current goal is `B`, replaces the goal current by A.
- `~A` and if the current goal is `False`, replaces the current goal by A.
- `forall x: T, A x` and if the current goal is B, try to unify the proposition `A x` with `B` (by finding the value of x that matches), and solves the current goal.

The apply tactic repeats these operations in turn.

example :

`H: forall x y: nat, P x -> Q y -> R x y`

if the goal is `R 2 3`, then `apply H` replaces the current goal by the two sub-goals `P 2` and `Q 3`.

Remark : The unification algorithm used by apply is incomplete, `apply` fail to find a value for some variables in `H`. The variant : `apply H with (x1 := M1) ... (xn := Mn)` explicitly instantiates the variables.

Rewriting & Simplification

`simpl`

Tries to reduce a term (in the sense of the system's calculation rules) which constitutes the current sub-goal.

example :

- 1+1 becomes 2
- `1 + m = S (0 + m)` becomes `S m = S m`

`simpl` applies some heuristics to keep the result still readable instead of fully normalizing it.

`rewrite`

`rewrite H` (where `H` is the name of a hypothesis of the context) applies one of the following rules :

When `H` is of the form :

- `M1=M2` replaces in the current goal all occurrences of `M1` with `M2`

— `H2 -> M1=M2` replaces in the current goal all occurrences of `M1` with `M2` and adds a new subgoal for `H2`

Variant : `rewrite <-` replaces in the current goal all occurrences of `M2` with `M1`

`replace`
`replace t1 with t2` (where `t1` and `t2` are terms) replaces all *free occurrences* of `t1` from in the current goal with `t1`

— It generates an equality `t1=t2` as a subgoal.
— This subgoal is automatically solved if it occurs among the hypotheses, (or if its symmetric form occurs)

`fold` & `unfold`
The `unfold id` tactic replaces for the current goal all occurrences of the identifier `id` with the body of its definition in the current environment.

example : `unfold andb`.
`andb a b` → (`if a then b else false`)

The `fold id` tactic does the inverse : it replaces every occurrence of the resulting terms in the selected hypotheses and/or goal will be replaced by its associated term.

example : `fold (andb a b)`.
(`if a then b else false`) → `andb a b`

Reasonning on Inductive Types

`induction`
`induction x` (where `x` is the name of a term whose type was declared using the **Inductive** command) applies mathematical induction on the current goal.

Given a goal `P(x)`, it generates two subgoals :

— `P(0)`
— `P(S n)` where a natural `n` and a hypothesis `IHn: P(n)` have been added to the context.

The variant `induction x using P` (where P is a proven Proposition of the form $p_0 \implies p_1 \implies \cdots \implies p_{n-1}$) does the same thing as `induction` except for :

Generates n subgoals : `P(0)`,`P(1)` … `P(n−1)` and adds a hypothesis `IHn: P(n)` to the context.

`discriminate`
`discriminate H` (where `H` is the name of a hypothesis of the context of the form `t1 = t2` with t1 and t2 starting with different constructors) solves the current goal.

`injection`
`injection H` (where `H` is the name of a hypothesis of the context of the form `t1 = t2` with t1 and t2 starting with different constructors) deduces equalities `u1 = u2`, `v1 = v2`, between corresponding subterms and add these equalities as new hypothese.

Solving Goals

`reflexivity`
Solves goals with the form `t = u`, when `t` and `u` are definitionally equal. Also solves the goal `True`.

`assumption`
Solves the current goal when there is a hypothesis in the local context whose type is convertible to the goal.

`contradiction`
`contradiction` tries to prove the current goal by finding a contradiction.

When the context contains :

— a pair of hypotheses where one is the negation of the other : `P` and `not P`
— a hypothesis with an empty inductive type (`e.g. False`),
— a hypothesis (`not P`) where `P` is a singleton inductive type (`e.g. True`).

Control Flow

`;` Operator
`tac1 ; tac2` (Where `tac1` and `tac2` are any tactic) runs `tac1` and then run `tac2` on all the subgoals created by `tac1`.

If after running `tac1` there is :

— no goal left then `tac2` is never run and Coq simply silently succeeds.
— one goal left then `tac1; tac2` is equivalent to `tac1. tac2`
— otherwise `tac2` is applied to all of the generated subgoals

`repeat`
The repeat loop `repeat tac` (Where `tac` is any tactic) repeats the tactic `tac` until it fails or leaves the proof context unchanged.

`try`
`try tac` (Where `tac` is any tactic) applies `tac` to each focused goal independently. If the application fails in a goal, it catches the error and leaves the goal unchanged.

Misc

`in`
The *localization* tactic `in`, has as a syntax : `tactic in (ident+) *?`

Where `ident` is a name in the context.

— On the left side of in, tactic can be `apply`, `unfold`, `rewrite` …
— The operation described by the tactic is performed in the facts listed after `in` and in the goal if a ∗ ends the list of names.

example :
`unfold not in *` replaces the (`not P`) with (`P -> False`) in the goal and all hypotheses.

`assert`
`assert P` (where P is any proposition of type Prop) adds a new hypothesis `H` to the current subgoal and a new subgoal before it to prove `H`.

`absurd`
`absurd P` (where P is any proposition of type Prop) deduces the current goal from `False`, and generates as subgoals `not P` and `P`. This is the ⊥-elimination of natural deduction.