Coq Tactics overview

Semantics 2023/2024

Based on material from the Introduction to Computational Logic Course at Saarland University

We write down goals using inference rules of the form

$$\frac{\text{premise}}{\text{conclusion}}$$

In Coq, this means that the premise is part of your proof context ("what you know ") while the conclusion is your goal ("what you have to prove").

The common variants refer to the table further down below.

Tactic	Applies to goals of the form	Goals after application	Proof term
destruct x Performs a case analysis on members x of inductive types X. • Use *eqn: H* to preserve the equations for the cases. • Common variants: as, in	$\frac{}{\forall \mathsf{x}\colon \mathsf{X},\ \mathsf{p}\ \mathsf{x}}$ where X is an inductive type.	depends on the inductive definition	λ (x: X) ⇒ match x with | C1 x1 ... xn ⇒ ■ | ... end
edestruct x Similar to **destruct**, but can also deal with existential variables: if it does not know how to instantiate variables, it does not fail, but instead introduces existential variables which need to be instantiated later.			

Tactic	Applies to goals of the form	Goals after application	Proof term
reflexivity Solves equalities that hold by conversion.	$\overline{\mathsf{x} = \mathsf{y}}$ where x and y are convertible	Solved. (fails if unsolvable)	Q
simpl Computes, in the sense that it reduces matches and fixpoints when applied to a constructor. simpl [X] to only use delta conversion on X. simpl −[X] to use delta conversion on everything except X. Common variants: in	$\overline{\mathsf{p\ (S\ x + y)}}$	$\overline{\mathsf{p\ (S\ (x + y))}}$	(unchanged)
rewrite H Replaces every occurrence of the term s with the term t if H: s = t (if H contains a universal quantifier, Coq might replace only one occurrence). • `rewrite !H` rewrites as often as possible, but at least one time. • specify the rewrite direction with *rewrite* → or *rewrite* ← • Common variants: in, at	$\dfrac{\mathsf{H{:}\ s = t}}{\mathsf{P}}$ where s occurs in P	$\dfrac{\mathsf{H{:}\ s = t}}{\mathsf{P'}}$ where s is substituted with t	*eq_indr* _ ■ *H*

Tactic	Applies to goals of the form	Goals after application	Proof term
induction x as [...] using ... in ... Applies the automatically generated induction lemma for an inductive type X to the current goal and introduces assumptions. • x can be number n. In that case assumptions until the nth term are introduced and an induction on the last introduced assumption is performed. • The keyword using can be used to perform induction with the eliminator provided. • An occurrence set with `in y |- *` can be used to generalize y in the inductive hypothesis similar to revert. • Common variants: as	$\dfrac{}{\forall\, x{:}\ X,\ p\ x}$ where X is an inductive type.	depends on the inductive definition	E_X ■ ... ■
cbv call by value - reduces the goal with specified reduction. Reductions: beta, delta (can have an identifier), match, fix, zeta If used without an argument, it reduces to a normal form Common variants: in	$\dfrac{}{P}$	$\dfrac{}{Q}$ with $P \approx Q$	(unchanged)
exact E Inserts the proof term E.		Solved. (fails if unsolvable)	E
assumption Solves the goal if the claim is an assumption.	$\dfrac{H : P}{P}$	Solved. (fails if unsolvable)	H

Tactic	Applies to goals of the form	Goals after application	Proof term
intros x1 xn H1 Hn Introduces the variables x1, xn, H1, Hn from the claim as an assumption. If used without arguments, the names of the assumptions are chosen. See the section Intro Patterns below for introduction patterns.	$\overline{\forall \text{ x1 x2, P} \to \text{Q} \to \text{R}}$	x1 : X xn : Y H1 : P Hn : Q $\overline{}$ R	(λ x1 xn H1 Hn $\Rightarrow$ ■), where ■ is the gap in the proof term which still needs to be closed
apply H Applies the function *H*. • apply H with (x:= a) applies H where x is instantiated with a. • Common variants: as, in	$\frac{\text{H: X} \to \text{Y}}{\text{Y}}$	$\frac{\text{H: X} \to \text{Y}}{\text{X}}$	*H* ■
eapply H Behaves like apply but does not fail when it can't instantiate variables. It can introduce existential variables. These need to be instantiated later. Common variants: as, in	$\frac{\text{H: X} \to \text{Y}}{\text{Y}}$	$\frac{\text{H: X} \to \text{Y}}{\text{X}}$	*H* ■
left. Applies the *L* contructor of $\vee$.[1]	$\overline{\text{P}\vee\text{Q}}$	$\overline{\text{P}}$	*L* ■
right. Applies the *R* contructor of $\vee$.[1]	$\overline{\text{P}\vee\text{Q}}$	$\overline{\text{Q}}$	*R* ■

[1] applies to types with two constructors

Tactic	Applies to goals of the form	Goals after application	Proof term
split Applies the C contructor of $\land$.[2]	$\overline{\mathsf{P \land Q}}$	$\overline{\mathsf{P}}$ $\quad$ $\overline{\mathsf{Q}}$	$C\ \blacksquare\ \ \blacksquare$
enough (H: Q) Allows you to prove P under the assumption H first and then H remains to be shown. Common variants: by, as	$\overline{\mathsf{P}}$	$\dfrac{\mathsf{H: Q}}{\mathsf{P}}$ $\quad$ $\overline{\mathsf{Q}}$	$(\lambda\ H \Rightarrow \blacksquare\ )\ \ \blacksquare$
exfalso Changes the goal to $\bot$ as it is always sufficient to prove falsity.	$\overline{\mathsf{P}}$	$\overline{\bot}$	$\bot\_\mathit{ind}\ P\ \ \blacksquare$ alternative: *match* F *[]* where F is a proof of $\bot$.
refine s Applies the partial proof term s, in the sense that for every underscore in the proof term a goal will be generated.	$\overline{\mathsf{P}}$	$\overline{\mathsf{Q}_1}$ $\quad$ $\overline{\mathsf{Q}_n}$ where $Q_1, \ldots, Q_n$ are the types of the underscores in s	$s_{\blacksquare}$
change Q Changes the goal to Q if P and Q are equal up to conversion (P $\approx$ Q). • The variant **change** P **with** Q can be used to replace a particular subterm P with Q. • Common variants: in, at (only together with **with**)	$\overline{\mathsf{P}}$	$\overline{\mathsf{Q}}$ with $P \approx Q$	(unchanged)
pattern x Performs a β-expansion on the goal. Common variants: at	$\overline{\mathsf{p\ x}}$	$\overline{(\lambda\ \mathsf{y} \Rightarrow \mathsf{p\ y})\ \mathsf{x}}$	(unchanged)

[2] applies to single constructor types

Tactic	Applies to goals of the form	Goals after application	Proof term
assert (H: Q) Asserts that the proposition P holds and makes it available as H in the further proof. Common variants: by, as	$\overline{\mathsf{P}}$	$\overline{\mathsf{Q}}$ $\quad \dfrac{\mathsf{H: Q}}{\mathsf{P}}$	*let* $H :=$ ■ *in* ■ alternative: $(\lambda\ H \Rightarrow$ ■ $)$ ■
fold H Undo the effect of delta reduction on H.	$\overline{\mathsf{P}}$	$\overline{\mathsf{Q}}$ with $P \approx Q$	(unchanged)
set (H := t) replaces t with H and adds a new definition H := t.	$\overline{\mathsf{P}}$	$\dfrac{\mathsf{H := t}}{\mathsf{Q}}$ where t is replaced by H in Q.	*let* $H := t$ *in* ■
subst x Replaces x with an equivalent value defined by an equation involving x in the assumptions or a definition of x. After that the assumption or definition is removed. **subst** without any arguments substitutes everything it can.	$\dfrac{\mathsf{x := t}}{\mathsf{p\ x}}$	$\overline{\mathsf{p\ t}}$	(unchanged)
revert xn x1 Hn H1 Reverse operation of introduce. Reverts the variables x1, xn, H1, Hn from the assumptions into the claim. The variabled must not be used in other assumptions.	x1 : X xn : Y H1 : P Hn : Q $\overline{}$ R	$\overline{\forall\ \mathsf{xn\ x1,\ Q \rightarrow P \rightarrow R}}$	(λ x1 xn H1 Hn $\Rightarrow$ ■ xn x1 Hn H1), where ■ is the gap in the proof term which still needs to be closed

Tactic	Applies to goals of the form	Goals after application	Proof term
replace A **with** B Replaces all occurrences of A with B in the goal. A new subgoal of the form A= B is generated and solved if it occurs in the assumptions. Common variants: in, by	$\overline{\mathsf{P}}$	$\overline{\mathsf{Q}}$ $\quad$ $\overline{A{=}B}$ where A is replaced by B in Q	$R \blacksquare \blacksquare \blacksquare$
exists a Puts in a witness into a proof with an existential quantifier.	$\overline{\exists\, \mathsf{x},\ \mathsf{p\ x}}$	$\overline{\mathsf{p\ a}}$	$E \blacksquare \blacksquare \blacksquare$
enough (H: Q) Asserts that the proposition P holds and makes it available as H in the further proof like assert. The difference is that the old goal is the first to prove.	$\overline{\mathsf{P}}$	$\dfrac{\mathsf{H{:}\ Q}}{\mathsf{P}}$ $\quad$ $\overline{\mathsf{Q}}$	*let* $H :=$ $\blacksquare$ *in* $\blacksquare$ alternative: $(\lambda\ H \Rightarrow \blacksquare\ )\ \blacksquare$
unfold f Unfolds the definition of f and β-reduces (reducing a λ applied to some argument) the result, if possible.	$\overline{\mathsf{P}}$ where P contains f somewhere	$\overline{\mathsf{P}'}$ where P' is P with the definition of f substituted in	(unchanged)
specialize (H x) Instantiates an assumption H by passing it an argument x. Common variants: as	$\dfrac{\mathsf{H{:}\ \forall\, y,\ P}}{\mathsf{Q}}$	$\dfrac{\mathsf{H{:}\ P}^{x}_{y}}{\mathsf{Q}}$	$(\lambda\ (H{:}\ P\,^{x}_{y} \Rightarrow \blacksquare$ $)\ (H\ x)$

Tactic	Applies to goals of the form	Goals after application	Proof term
clear H Removes a hypothesis.	$\dfrac{\text{H: X} \to \text{Y}}{\text{Y}}$	$\overline{\text{X}}$	■
contradict H Changes the goal to $\bot$ and apply the assumption H if it is a negation. When the goal is a negation itself it first introduce it. When H is not an implication to $\bot$ it changes the goal to the negation of H.	$\dfrac{\text{H: X}}{\text{Y}}$	$\overline{\text{X}}$	$\bot_{ind}\ Y\ H$ ■
f_equal Functions are functional, thus if we want to show $f\ x = f\ y$ it's always sufficient to show $x = y$. This is also true for constructors by injectivity.	$\overline{\text{f x1 xn = f y1 yn}}$	$\overline{\text{x1 = y1}}$ $\overline{\text{xn = yn}}$	f_equal ■ ... ■
symmetry Applies symmetry to equalities. Common variants: in	$\overline{\text{x= y}}$	$\overline{\text{y= x}}$	eq_sym _
constructor [n] Applies the nth constructor to the goal. If no number is specified the constructors are tried in order. It is a more general form of the tactics split , ∃ , left and right .	$\overline{\text{P}}$	$\overline{\text{Q}}$	C _ where C is the constructor
generalize H Add universal quantification of H to the Goal. This might manifest in an implication if H is not used in the previous goal.	$\overline{\text{P}}$	$\overline{\forall \text{H. P}}$	

Tactic	Applies to goals of the form	Goals after application	Proof term
decide equality Construct an equality decider for inductive types. The constructors can't have proofs, functions nor objects in dependent types.	$\frac{}{\forall\, x\, y : R,\ \{x = y\} + \{x \neq y\}}$	$\frac{H}{Q}$	
try t Use tactic t if possible. Does nothing otherwise.	$\frac{}{P}$	$\frac{}{Q}$	

Automation

Tactic	Applies to goals of the form	Goals after application
tauto Solves all goals that can be solved by purely propositional reasoning. It can solve all tautological intuitionistic propositions. `tauto` will not instantiate universal quantifiers.	$\overline{\mathsf{P}}$	Solved. (fails if unsolvable)
auto `auto` tries the assumptions, then introduce and tries tactics depending on the form of the goal A number can be used to define the search depth. Its proof search can be customised by adding hints.	$\overline{\mathsf{P}}$	Solved. (does nothing if unsolvable)
eauto A more general tactic than auto. It can resolve existential quantifiers. Leave Variable which can't be instantiated as existential variables.	$\overline{\mathsf{P}}$	Solved. (does nothing if unsolvable)
congruence Solves all goals that can be solved using purely equational reasoning, i.e reflexivity, transitvity, symmetry and rewriting. It uses the Nelson and Oppen closure algorithm. It subsumes the power of injectivity and discriminate.	$\overline{\mathsf{P}}$	Solved. (fails if unsolvable)
discriminate Can prove anything when a disjoint assumption is present. It automates a disjointness proof of constructors.	$\overline{\mathsf{P}}$	Solved. (fails if unsolvable)

lia Uses linear positivstellensatz refutations, cutting plane proofs (rounding rational constants) and case analysis for possible values. Has the power of omega (Presburger Arithmetic) and normalization of ring and semiring structures. Put simply, it solves arithmetic computational problems. Lia has to be loaded before (`Require Import Lia.`).	$\overline{\mathsf{P}}$	Solved. (fails if unsolvable)
nia A variant of lia that can not only deal with linear arithmetic, but also with non-linear arithmetic (i.e. multiplication). Essentially heuristically transforms the goal to eliminate non-linearities and then calls lia. This is not a complete decision procedure and may fail on many goals or take prohibitely long. Lia has to be loaded before (`Require Import Lia.`).	$\overline{\mathsf{P}}$	Solved. (fails if unsolvable)
intuition Split along the search tree of the decision procedures from tauto and apply auto.	$\overline{\mathsf{P}}$	Solved. (simplify if unsolvable)
assumption Can use hypothesis which type is convertible to the goal to proof it.	$\overline{\mathsf{P}}$	Solved. (fail if unsolvable)
eassumption Behave like assumption but can handle goals with existential variables.	$\overline{\mathsf{P}}$	Solved. (fail if unsolvable)
firstorder Uses logical connectives and first-order class inductive definitions to solve problems of predicate logic.	$\overline{\mathsf{P}}$	Solved. (does not fail)
trivial This is a restricted version of auto that is not recursive. Essentially combines reflexivity and assumption.		Solved. (does not fail)

injection H Injectivity proofs of constructors	$\dfrac{\text{H: C x1 xn} = \text{C y1 yn}}{\mathsf{P}}$	$\dfrac{}{\mathsf{x1} = \mathsf{y1} \rightarrow \mathsf{xn} = \mathsf{yn} \rightarrow \mathsf{P}}$	
discriminate H Disjointness proofs of constructors	$\dfrac{\text{H: C1 x1 xn} = \text{C2 y1 ym}}{\mathsf{P}}$	Solved. (fails if unsolvable)	

Essential stdpp tactics

Tactic	Applies to goals of the form	Goals after application
done Solves trivial goals by reflexivity, discrimination, splitting, and with trivial. Faster than Coq's built-in easy.	$\overline{\mathsf{P}}$	Solved. (fails if unsolvable)
simplify_eq Repeatedly substitutes, discriminates, and injects equalities, and tries to contradict impossible inequalities. The variant simplify_eq/= additionally performs simplification.	$\overline{\mathsf{P}}$	Simplified goal.
by tac Calls tac and executes done afterwards. Faster than Coq's built-in now.	$\overline{\mathsf{P}}$	Solved (fails if unsolvable).
split_and Destructs a conjunction in the goal (and only conjunctions, in contrast to Coq's built-in split, which also splits other inductives). The variant split_andsplits multiple conjunctions, but at least one. The variant split_and? splits zero or more conjunctions.	$\overline{\mathsf{P} \wedge \mathsf{Q}}$	$\overline{\mathsf{P}}$ $\overline{\mathsf{Q}}$
naive_solver A firstorder-like tactic. firstorder can "loop" on quite small goals already, naive_solver fixes that by implementing a breadth-first search with limited depth. It implements some ad-hoc rules for logical connectives that in practice work quite well, and usually works better than firstorder for our purposes.	$\overline{\mathsf{P}}$	Solved (fails if unsolvable).

Common variants of tactics

This is a small list of common variants of tactics (e.g. `apply` has a variant `apply _ in _`) together with the behaviour one can in general expect from them. However, there may be minor differences in the effect of these variants depending on the tactic, although Coq mostly tries to make them behave as expected.

However, exceptions confirm the rule. Sometimes the same keywords can mean very different things (take for instance the variant induction ... in ... mentioned above).

The tactic list above contains for each tactic a list of the most relevant variants. For a more comprehensive list of variants and a description of their individual behaviours, see the Coq tactic index.

Variant	Usual Meaning	Example
as `<intropattern>`	Use an intropattern to specify the names given to new assumptions introduced or to directly destruct it.	destruct H as [H1 H2] apply H in H' as [H1 H2]
by `<tactical>`	Directly dispatch a new goal that is generated by a given tactical, which should completely solve the goal.	assert (x = y) by (intros H; now apply H2) rewrite H by eauto
in `<assumption>`	If a tactic should not be applied to the goal, specify to which assumption it should be applied.	rewrite H in H1 apply H1 in H2
at `<occurrence list>`	For rewriting-based tactics: give the occurrence (s) at which the rewrite shall be performed.	rewrite H at 1 3 change y at 2 with ((fun x => x) y)

How to use tactics

Proposition	Usage	Proving
A $\to$ B	If you want to prove B you can apply it to change your goal into A.	You introduce (intros) A as a new assumption and are left with B to prove.
$\neg$ A	When you have an instance of A you can construct $\perp$.	Introduce A and proof $\perp$.
$\forall x, px$	You can apply your assumption with a specified parameter y to prove p y.	Introduce (intros) an arbitrary x and prove p x.
A $\wedge$ B	By case analysis (destruct) you have an assumption of type A and another one of type B.	You can split the goal into two subgoals, where the first is A and the second is B.
A $\vee$ B	By case analysis (destruct) you have either A or B. There will be two subgoals. In the first one, you have A as an additional assumption and in the second one you have B as an additional assumption.	Decide to either prove the left side A (left) or the right side B (right).

Auxiliary commands

COMMAND	USAGE
Print	displays information about objects like the definition of a function or the proof of a theorem. Print All shows the current state of the environment.
Locate	shows information about a given notation
Compute	evaluate a given term with call-by-value
Check	displays the type of a given term
Arguments	syntax: Arguments qualifier [n] {m} where n and m are arguments arguments in square brackets become implicit and arguments in curly brackets are declared as maximally inserted.
Show Proof	displays the proof term which has been constructed by tactics
Section [Name]	open a new section with local declarations. Outside the section all used declared constants have to be provided when accessing the object from the section.
Variable	Assume a constant of a given type inside a section.
Module Type [Name]	create a type for modules with objects given by Parameter X:T.
Module [Name] : [Type]	creates a module given a module type. All Parameters have to be defined. Objects which are not specified by a parameter are not visible to the outside.
Require Import [Name]	Loads and declares the module. After this the content of the module is imported.
if x then a else b	Performs a match on (x:X) where X has two constructors. The statement in the first case is a and the statement for the second constructor is b. All parameters of the constructors are ignored.
let $(y_1, \ldots, y_n)$:= x in a	Performs a match on (x:X) where X has one constructor. y_1 to y_n are bound to the last n parameters of the constructor, where y_n is the last parameter. a is the statement in the match.
Proof. ... Admitted.	End an unsolved proof.
Proof. ... Abort.	End an unsolved proof and remove it from the context.
Proof. ... Qed.	Extracts a proof term as justification for the goal and declares the goal as opaque constant. An opaque constant can't be unfolded and so can't be used in computational evaluation.
Proof. ... Defined.	Declares the proof as transparent. It can be unfolded by conversion and so the proof term can be used explicitly.
Search (P).	Print the lemma which matches a provided pattern P. Example for commutativity: (_ + _ = _ + _)

Command	Usage
Fail t.	Execute t and expects it to fail. If it doesn't the command fail.
all: t.	Execute t in all parallel goals.
n: t.	Execute t in the nth subgoal.
@H	Explicitly takes implicit arguments.
Print Assumptions f.	Show all assumptions (axioms, hypothesis, variables, ...) made to define f.

Intro Patterns

These patterns can be used together with tactics that introduce new assumptions, like `intros`.

PATTERN	DESCRIPTION	BEFORE	AFTER
Name	Introduce a term (hypothesis or variable) as Name	$\overline{\mathsf{H} \to \mathsf{P}}$	$\dfrac{\mathsf{Name{:}\ H}}{\mathsf{P}}$
[x|n H1 H2]	Introduce a term and destruct it. If the type has multiple constructors they are separated by |. The arguments in one constructor are separated by spaces.	$\overline{(\exists\, \mathsf{k},\ \mathsf{H}) \to \mathsf{P}}$	$\mathsf{k} : \mathbb{N}$ $\mathsf{H1} : \mathsf{H}$ $\overline{}$ P
*	Introduce one or more quantified variables until there are no more quantified variables.	$\overline{\forall\ \mathsf{H1}\ \mathsf{H2},\ \mathsf{H3} \to \mathsf{P}}$	$\dfrac{\mathsf{n1} : \mathsf{H1},\ \mathsf{n2} : \mathsf{H2}}{\mathsf{H3} \to \mathsf{P}}$
H1%H	Introduce a term and apply H in it.	$\mathsf{H} : \mathsf{X} \to \mathsf{Y}$ $\overline{}$ $\mathsf{X} \to \mathsf{P}$	$\mathsf{H} : \mathsf{X} \to \mathsf{Y}$ $\mathsf{H1} : \mathsf{Y}$ $\overline{}$ P
$\to$	Rewrite the equation. Can also be used as $\leftarrow$.	$\overline{(\mathsf{X}{=}\ \mathsf{Y}) \to \mathsf{Y}}$	$\overline{\mathsf{X}}$

Pattern	Description	Before	After
(H1&H2&H3)	Introduce a nested term with multiple arguments and split it.	$\overline{}$ $\mathsf{X} \wedge \mathsf{Y} \wedge \mathsf{Z} \to \mathsf{P}$	$\mathsf{H1} : \mathsf{X}$ $\mathsf{H2} : \mathsf{Y}$ $\mathsf{H3} : \mathsf{Z}$ $\overline{}$ P
[= H]	Introduce a term and apply injectivity and discriminate on it.	$\overline{}$ $\mathsf{S}\ \mathsf{x} = \mathsf{S}\ \mathsf{y} \to \mathsf{P}$	$\mathsf{H} : \mathsf{x} = \mathsf{y}$ $\overline{}$ P