Coq チートシート

記号	使用前		⟹	使用後	
	仮定	ゴール	tactic	仮定	ゴール
仮定	p:P	P	exact p.		証明終わり
→		P -> Q	intro p0.	p0:P	Q
->	pq: P -> Q	Q	apply pq.	pq0: P -> Q	P
∧		P /\ Q	split.		(1/2) P (2/2) Q
/\	pq: P /\ Q		destruct pq as [p0 q0].	p0:P q0:Q	
∨		P \/ Q	left.		P
			right.		Q
\/	pq:P \/ Q		destruct pq as [p0|q0].	(1/2) p0:P (2/2) q0:Q	
⊤		True	exact I.		証明終わり
⊥	H:False		elim H.		証明終わり
¬		~P	intro p0.	p0:P	False
~	np:~P		elim np.	np:~P	P
∀		forall x:X, f x y = ...	intro x0.	x0:X	f x0 y = ...
forall	H:forall x:X, f x = ...	f x0 = ...	exact (H x0).		証明終わり
∃	x0:X	exists x:X, f x y = ...	exists x0.	x0:X	f x0 y = ...
exists	H:exists x:X, f x = ...		destruct H as [x0 H0].	x0:X H0:f x0 = ...	
$\beta\iota$ 簡約		簡約したい式	simpl.		簡約された式
等式		x = x	reflexivity.		証明終わり
書き換え	H:foo = bar	f foo = g bar	rewrite H.	H:foo = bar	f bar = g bar
	H:foo = bar	f foo = g bar	rewrite <- H.	H:foo = bar	f foo = g foo
		f foo = g bar	replace foo with baz.		(1/2) f baz = g bar (2/2) baz = foo
構築子	H:C a = C b	P	injection H.	H:C a = C b	a=b -> P
		C a = C b	f_equal.		a=b
	H:C1 .. = C2 ..		discriminate H.		証明終わり

名前の自動生成 `intro. destruct x.` などで自動的に名前を生成してくれるが、自分で名前を付ける方が良い習慣。

intros `intros p q.` で複数の `intro` を代用可能。

対象 `simpl, rewrite, replace` は `simpl in H.` や `simpl in *.` など tactic 対象を変更出来る。

場合分け `destruct x.` で `x` の (構築子毎の) 場合分けが出来る。似たものとして `case_eq x.` もある。

帰納法 `induction x.` で `x` の 帰納法が出来る。

generalize 仮定 `x:X` がある時、`generalize dependent x.` するとゴールが `forall x:X, ...` になる。

inversion `injection, discriminate` を纏めたような強力な tactic。仮定に対する場合分けが出来る。

e 系 tactic `eauto, eapply, erewrite` などは定理名から引数を推測するので省略可能。

assert 証明の途中で補題を作りたい時に `assert(H:hogehoge).` とすると、ゴールが `hogehoge` に切り替わる。証明が終わると `H:hogehoge` が仮定に追加される。

remember `remember (f foo bar) as x.` とすると、仮定 `Heqx:x = f foo bar` が追加され、`f foo bar` が `x` で置き換えられる。`remember` して `destruct, induction` することもあり。

info `info auto.` とすると、`auto.` の実行した中身を調べられる。(Coq 8.3 まで。)

fold/unfold `Definition f := ...` で定義した定義を展開するのは `unfold f` で、逆が `fold f`。`unfold f; fold f.` で元に戻るのではなく、いい感じに簡約されることがある。

ring `Require Import Ring.` すると使える自動証明。`ring.` で環の等式 (加減算と乗算に関する等式) を解く。

omega `Require Import Omega.` すると使える自動証明。`omega.` で一次式に関する等式不等式 (正確には Presburger 算術の式) を解く。