Logic in Coq (Logic.v)

Keywords

- `Prop : Type` for propositions
- `Axiom`
- `Print Assumptions` : to check whether a particular proof relies on any additional axioms

Tactics

- `split` : split the *goal* `A /\ B` into two subgoals `A` and `B`
- `destruct (A /\ B) as [HA HB].` : remove `A /\ B` from the context and add two new hypotheses `HA` , stating that `A` is true, and `HB` , stating that `B` is true
- `intros [HA HB].` implicitly destruct `A /\ B`
- `intros [HP [HQ HR]].`
- `intros [Hn | Hm].` implicitly destruct `A \/ B` in the *context*
- `left.` `right.` to prove the left side or the right side or a disjunction in the *goal*
- `apply H in L as [A B].` `apply` and `destruct`
- `destruct H.` ***where `H` is `False`***
- `exfalso.` : `apply ex_falso_quodlibet.` If you are trying to prove a *goal* that is nonsensical, apply `ex_falso_quodlibet` to change the goal to `False` .
- `apply I.` `I : True.`
- `rewrite -> H.` `H` can be the form of `A <-> B`
- `reflexivity H.` `H` can be the form of `A <-> B`
- `apply H.` `H` can be the form of `A <-> B`
- `exists t.` : to prove `exists x : ...` in the *goal*
- `intros [m Hm].` for `(exists m, n = m + 4) ->` : implicitly destruct `exists x, P` in the context to obtain a witness `x` and hypothesis stating that `P` holds of `x`
- `rewrite (plus_comm y z).` apply `plus_comm` to the arguments we want to instantiate with, in much the same way as we apply a polymorphic function to a type argument
- `apply in_not_nil with (x := 42).`

- `apply in_not_nil in H.`
- `apply (in_not_nil 42).` apply lemma `in_not_nil` to the value for `x` (`x := 42`)
- `apply (in_not_nil _ _ _ H).` apply lemma `in_not_nil` to a hypothesis

Defintions and Theorems

`Prop`

- `Definition injective {A B : Type} (f : A -> B) := forall x y : A, f x = f y -> x = y.`
- `Lemma succ_inj : injective S.`

Logical Connectives

Conjunction

- `Lemma and_intro : forall A B : Prop, A -> B -> A /\ B.`
- `Example and_exercise : forall n m : nat, n + m = 0 -> n = 0 /\ m = 0.`
- `Lemma proj1 : forall P Q : Prop, P /\ Q -> P.`
- `Lemma proj2 : forall P Q : Prop, P /\ Q -> Q.`
- `Theorem and_commut : forall P Q : Prop, P /\ Q -> Q /\ P.`
- `Theorem and_assoc : forall P Q : Prop, P /\ (Q /\ R) -> (P /\ Q) /\ R.`

Disjunction

- `Lemma or_example : forall n m : nat, n = 0 \/ m = 0 -> n * m = 0.`
- `Lemma or_intro : forall A B : Prop, A -> A \/ B.`
- `Lemma zero_or_succ : forall n : nat, n = 0 \/ n = S (pred n).`
- `Lemma mult_eq_0 : forall n m, n * m = 0 -> n = 0 \/ m = 0.`
- `Theorem or_commut : forall P Q : Prop, P \/ Q -> Q \/ P.`

Falsehood and Negation

- `Module MyNot.`

- `Definition not (P : Prop) := P -> False.`
- `Notation "~ x" := (not x) : type_scope.`
- `End MyNot.`
- `Theorem ex_falso_quodlibet : forall (P : Prop), False -> P.`
- `Fact not_implies_our_not : forall (P:Prop), ~ P -> (forall (Q:Prop), P -> Q).`
- `Notation "x <> y := (~ (x = y))."`
- `Theorem zero_not_one : 0 <> 1.`
- `Theorem contradiction_implies_anything : forall P Q : Prop, (P /\ ~P) -> Q.`
- `Theorem double_neg : forall P : Prop, P -> ~~P.`
- `Theorem contrapositive : forall (P Q : Prop), (P -> Q) -> (~Q -> ~P).`
- `Theorem not_both_true_and_false : forall P : Prop, ~ (P /\ ~P).`
- `Theorem not_true_is_false : forall b : bool, b <> true -> b = false.`

Truth

- `Lemma True_is_true : True.`

Logical Equivalence

- `Module MyIff.`
- `Definition iff (P Q : Prop) := (P -> Q) /\ (Q -> P).`
- `Notation "P <-> Q := (iff P Q)."`
- `End MyIff.`
- `Theorem iff_sym : forall P Q : Prop, (P <-> Q) -> (Q <-> P).`
- `Lemma not_true_iff_false : forall b : bool, b <> true <-> b = false.`
- `Theorem iff_refl : forall P : Prop, P <-> P.`
- `Theorem iff_trans : forall P Q R : Prop, (P <-> Q) -> (Q <-> R) -> (P <-> R).`
- `Theorem or_distributes_over_and : forall P Q R : Prop, P \/ (Q /\ R) <-> (P \/ Q) /\ (P \/ R).`
- `Lemma mult_0 : forall n m, n * m = 0 <-> n = 0 \/ m = 0.`
- `lemma or_assoc : forall P Q R : Prop, P \/ (Q \/ R) <-> (P \/ Q) \/ R.`

Existential Quantification

- `Theorem dist_not_exists : forall (X : Type) (P : X -> Prop), (forall x , P x) -> ~ (exists x, ~ P x).`
- `Theorem dist_exists_or : forall (X:Type) (P Q : X -> Prop), (exists x, P x \/ Q x) <-> (exists x, P x) \/ (exists x, Q x).`

Programming with Propositions

- `Fixpoint In {A : Type} (x : A) (l : list A) : Prop`
- `Lemma In_map : forall (A B : Type) (f : A -> B) (l : list A) (x : A), In x l -> In (f x) (map f l).`
- `Theorem In_map_iff : forall (A B : Type) (f : A -> B) (l : list A) (y : B), In y (map f l) <-> exists x, f x = y /\ In x l.`
- `Theorem In_app_iff : forall A l l' (a : A), In a (l ++ l') <-> In a l \/ In a l'.`
- `Lemma All_In : forall T (P : T -> Prop) (l : list T), (forall x, In x l -> P x) <-> All P l.`
- `Definition combine_odd_even (Podd Peven : nat -> Prop) : nat -> Prop`
- `Theorem combine_odd_even_intro : forall (Podd Peven : nat -> Prop) (n : nat), (oddb n = true -> Podd n) -> (oddb n = false -> Peven n) -> combine_odd_even Podd Peven n.`
- `Theorem combine_odd_even_elim_odd : forall (Podd Peven : nat -> Prop) (n : nat), combine_odd_even Podd Peven n -> oddb n = true -> Podd n.`
- `Theorem combine_odd_even_elim_even : forall (Podd Peven : nat -> Prop) (n : nat), combine_odd_even Podd Peven n -> oddb n = false -> Peven n.`
- `Lemma in_not_nil : forall A (x : A) (l : list A), In x l -> l <> [].`

Coq vs. Set Theory

- `Axiom functional_extensionality : forall {X Y : Type} {f g : X -> Y}, (forall (x : X), f x = g x) -> f = g.`

Propositions and Booleans

- `Theorem evenb_double : forall k, evenb (double k) = true.`
- `Theorem evenb_double_conv : forall n, exists k, n = if evenb n then double k else S (double k).`

- `Theorem even_bool_prop : forall n, evenb n = true <-> exists k, n = double k.`
- `Theorem eqb_eq : forall n1 n2 : nat, n1 =? n2 = true <-> n1 = n2.`
- `Theorem plus_eqb_example : forall n m p : nat, n =? m = true -> n + p =? m + p = true.`
- `Lemma andb_true_iff : forall b1 b2 : bool, b1 && b2 = true <-> b1 = true /\ b2 = true.`
- `Lemma orb_true_iff : forall b1 b2 : bool, b1 || b2 = true <-> b1 = true \/ b2 = true.`
- `Theorem eqb_neq : forall x y : nat, x =? y = false <-> x <> y.`
- `Lemma eqb_list_true_iff : forall A (eqb : A -> A -> bool), (forall a1 a2, eqb a1 a2 = true <-> a1 = a2) -> forall l1 l2, eqb_list eqb l1 l2 = true <-> l1 = l2.`
- `Fixpoint forallb {X : Type} (test : X -> bool) (l : list X) : bool`

Problems

- `Theorem In_map_iff` : simpler proof?
- `Lemma All_In` : simpler proof?
- `Lemma tr_rev_correct : forall X, @tr_rev X = @rev X.` How to prove it?
- `Lemma eqb_list_true_iff` : How to prove it?

> Written with [StackEdit](https://stackedit.io/).