More Basic Tactics (Tactics.v)

Keywords

Tactics

- `apply`
  - **Scenario 1:** the goal to be proved is *exactly* the same as some hypothesis in the context or some previously proved lemma
  - **Scenario 2:** `apply` *conditional* hypotheses and lemmas: if the statement *being applied* is an implication `A -> B` and the goal is `B`, then the premises `A` of this implication will be added to the list of subgoals needing to be proved. Typically, `H` in `apply H` begins with a `forall`
  - **Scenario 3:** `apply L in H`
  - **Scenario 4:** `apply trans_eq with (m := [c;d]).` or `apply trans_eq with ([c;d]).`
  - `apply` will perform `simpl` first, if needed
- `symmetry`:
  - `symmetry.`: switch the left and right sides of an equality in the *goal*
  - `symmetry in H.`: switch the left and right sides of an equality in `H`
- `injection`:
  - **Scenario 1:** `injection H.`: to generate all equations that can be inferred from `H` using the injectivity of constructors. Each such equation is added as a *premise* to the *goal*.
  - **Scenario 2:** `injection H as ...` choose names for the introduced equations; in this case, the introduced equations are automatically added into the *context*.
- `discriminate H`: "principle of explosion"
- `simpl in H.`

- `generalize dependent n.`
- `intros [m] [n].` in `forall x y : id,(Id m)`
- `unfold Definition.`
- `unfold Definition in H.`
- `destruct (Exp) eqn:E` destruct on compound expressions `Exp`

Definitions and Theorems

`apply`, `apply with`

- `Theorem trans_eq : forall (X : Type) (n m o : X), n = m -> m = o -> n = o.`

`injection` and `discriminate`

- `Theorem f_equal : forall (A B : Type) (f : A -> B) (x y : A), x = y -> f x = f y.`

Using Tactics on Hypotheses

- `Theorem S_inj : forall (n m : nat) (b : bool), (S n) =? (S m) = b -> n =? m = b.`
- `Theorem plus_n_n_injective : forall n m, n + n = m + m -> n = m.`
- `Theorem double_injective : forall n m, double n = double m -> n = m.`
- `Theorem eqb_true : forall n m, n =? m = true -> n = m.`
- `Theorem eqb_id_true : forall x y, eqb_id x y = true -> x = y.`
- `Theorem nth_error_after_last : forall (n : nat) (X : Type) (l : list X), length l = n -> nth_error l n = None.`

Unfolding Definitions

- `Definition square n := n * n.`
- `Lemma square_mult : forall n m, square (n * m) = square n * square m.`

Using `destruct` on Compound Expressions

- `Fixpoint split {X Y : Type} (l : list (X * Y)) : (list X) * (list Y)`
- `Theorem combine_split : forall X Y (l : list (X * Y)) l1 l2, split l = (l1, l2) -> combine l1 l2 = l.`
- `Theorem bool_fn_applied_thrice : forall (f : bool -> bool) (b : bool), f (f (f b)) = f b.`

Additional Exercises

- `Theorem eqb_sym : forall (n m : nat), (n =? m) = (m =? n).`
- `Theorem eqb_trans : forall n m p, n =? m = true -> m =? p = true -> n =? p = true.`
- `Definition split_combine_statement : Prop := forall (X Y : Type) (l1 : list X) (l2 : list Y), length l1 = length l2 -> split (combine l1 l2) = (l1, l2).`
- `Theorem filter_exercise : forall (X : Type) (test : X -> bool) (x : X) (l lf : list X), filter test l = x :: lf -> test x = true.`
- `Fixpoint forallb {X : Type} (test : X -> bool) (l : list X) : bool`
- `Fixpoint existsb {X : Type} (test : X -> bool) (l : list X) : bool`
- `Theorem existsb_existsb' : forall (X : Type) (test : X -> bool) (l : list X), existsb test l = existsb' test l.`

Problems

- `Theorem silly_ex` : what happens in `apply eq2.` ?
- `Theorem rev_exercise1` : simpler proof?
- `injection` without `in H` : for example, the goal is `[m] = [n].`
- `apply L in H` : why not `apply H in L`
- `Theorem plus_n_n_injective`
- `Theorem combine_split` : simpler proof?
- `length l1 = length l2 ->` vs. `length l1 =? length l2 = true ->`

- `Theorem split_combine` : `rewrite IHt.` How does it work?

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