Challenge

Spare Time Teaching

March 17, 2015

Problem

Many people who have worked with Coq knows that many tactics can use `eqn:`, tactics like `remember`, `case`, and `destruct`. However it doesn't work with `induction` which is annoying... and weird.

The challenge is to implement it (the `eqn:`-part should work like `case` and `destruct`) and use it to prove some simple lemmas.

```
Require Import Arith List.

Tactic Notation "better_induction" ident(var)
                                "as" simple_intropattern(p)
                              "eqn:" ident(H) :=
  ???

Lemma works_with_ℕ :
  ∀ n, n + 0 = n.
Proof.
  intro n.
  better_induction n as [ | n' IHn' ] eqn:H.
  ???
Qed.

Lemma works_with_lists :
  ∀ xs : list ℕ, xs ++ nil = xs.
Proof.
  intro xs.
  better_induction xs as [ | x xs' IHxs' ] eqn:H.
  ???
Qed.

Inductive tree :=
| Leaf : tree
| Node : tree → ℕ → tree → tree.

Fixpoint expand t1 t2 :=
```

```
  match t1 with
    | Leaf ⇒ t2
    | Node t1' n t2' ⇒ Node (expand t1' t2) n (expand t2' t2)
  end.

Lemma works_with_tree :
  ∀ t, expand t Leaf = t.
Proof.
  intros t.
  better_induction t as [ | t1 IHt1 n t2 IHt2 ] eqn:H.
  ???
Qed.
```