IndPrinciples: Induction Principles(`IndPrinciples.v`)

Keywords

Key Ideas

General rule for `t_ind`:

- The type declaration `t` gives several constructors; each corresponds to one clause of the induction principle.
- Each constructor `c` takes argument types `a1 ... an`.
- Each `ai` can be either `t` (the datatype we are defining) or some other type `s`.
- The corresponding case of the induction principle says:
  - "For all values `x1 ... xn` of types `a1 ... an`, if `P` holds for each of the inductive arguments (each `xi` of type `t`), then `P` holds for `c x1 ... xn`".

Tactics

- `apply nat_ind.` We do not introduce `n` into the context before applying `nat_ind`.
- ``

Definitions and Theorems

Basics

- `Theorem mult_0_r' : forall n : nat, n * 0 = 0.`
- `Theorem plus_one_r' : forall n : nat, n + 1 = S n.`
- `Inductive yesno : Type := | yes | no.`
- `Inductive rgb : Type := | red | green | blue.`

- `Inductive natlist : Type := | nnil | ncons (n : nat) (l : natlist).`
- `Inductive byntree : Type := | bempty | bleaf (yn : yesno) | nbranch (yn : yesno) (t1 t2 : byntree).`
- `Inductive ExSet : Type :=`

Polymorphism

- `Inductive list (X:Type) : Type := | nil : list X | cons : X → list X → list X.`
- `Inductive tree (X:Type) : Type := | leaf (x : X) | node (t1 t2 : tree X).`

Induction Hypotheses

Problems

- `Inductive foo (X : Type) (Y : Type) := | bar (x : X) | baz (y : Y) | quux (f1 : nat -> foo X Y). Check foo_ind.`
  如何生成 `quux` 对应的归纳原理 (Induction Principle)?

> Written with StackEdit.