Polymorphism and Higher-Order Functions (Poly.v)

Keywords

- `Arguments nil {X}.` : Implicit Arguments
- `Arguments cons {X} _ _.`
- `Arguments repeat {X} x count.`
- `Fixpoint repeat''' {X : Type} (x : X) (count : nat) : list X` : declare an argument to be implicit when declaring the function itself, by surrounding it in curly braces instead of parens.
- `Fail + any command` : this command indeed fails when executed
- `Check @nil.` : Force the implicit arguments to be explicit by prefixing the function name with `@`
- `fun n => n * n` : anonymous function

Tactics

Definitions and Theorems

Polymorphism

- `Inductive list (X : Type) : Type`
- `Fixpoint repeat (X : Type) (x : X) (count : Nat) : list X`
-
- `Module MumbleGrumble.`
- `Inductive mumble : Type`
- `Inductive grumble (X : Type) : Type`
- `End MumbleGrumble.`

Type Annotation Inference

- `Fixpoint repeat' X x count : list X`
- `Fixpoint repeat'' X x count : list X`
- `Fixpoint app {X : Type} (l1 l2 : list X) : list X`
- `Fixpoint rev {X : Type} (l : list X) : list X`
- `Fixpoint length {X : Type} (l : list X) : nat`
-
- `Notation "x :: y"`
- `Notation "[ ]"`
- `Notation "[ x ; .. ; y ]"`
- `Notation "x ++ y"`
-
- `Theorem app_nil_r : forall (X : Type), forall l : list X, l ++ [] = l.`
- `Theorem app_assoc : forall A (l m n : list A), l ++ m ++ n = (l ++ m) ++ n.`
- `Theorem app_length : forall (X : Type) (l1 l2 : list X), length (l1 ++ l2) = length l1 + length l2.`
- `Theorem rev_app_distr : forall (X : Type) (l1 l2 : list X), rev (l1 ++ l2) = rev l2 ++ rev l1.`
- `Theorem rev_involutive: forall (X : Type), forall (l : list X), rev (rev l) = l.`

Polymorphic Pairs (Products)

- `Inductive prod (X Y : Type) : Type`
- `Arguments pair {X} {Y} _ _.`
- `Notation "( x, y )" := (pair x y).`
- `Notation "X * Y" := (prod X Y) : type_scope.`
- `Definition fst {X Y : Type} (p : X * Y) : X`
- `Definition snd {X Y : Type} (p : X * Y) : Y`
- `Fixpoint combine {X Y : Type} (lx : list X) (ly : list Y) : list (X * Y)`
- `Fixpoint split {X Y : Type} (l : list (X * Y)) : (list X) * (list Y)`

Polymorphic Options

- `Module OptionPlayground`
- `Inductive option (X : Type) : Type`

- Arguments Some {X} _.
- Arguments None {X}.
- End OptionPlayground
- Fixpoint nth_error {X : Type} (l : list X) (n : nat) : option X
- Fixpoint hd_error {X : Type} (l : list X) : option X

Functions as Data

- Function doit3times {X : Type} (f : X -> X) (n : X) : X
- Fixpoint filter {X : Type} (test : X -> bool) (l : list X) : list X
- Definition length_is_1 {X : Type} (l : list X) : bool
- Definition partition {X : Type} (test : X -> bool) (l : list X) : (list X) * (list X)
- Fixpoint map {X Y : Type} (f : X -> Y) (l : list X) : (list Y)
- Theorem map_app_distr : forall (X Y : Type) (f : X -> Y) (l1 l2 : list X), map f (l1 ++ l2) = map f l1 ++ map f l2.
- Theorem map_rev : forall (X Y : Type) (f : X -> Y) (l : list X), map f (rev l) = rev (map f l).
- Fixpoint flat_map : {X Y : Type} (f : X -> list Y) (l : list X) : (list Y)
-
- Definition option_map {X Y : Type} (f : X -> Y) (xo : option X) : option Y
- Fixpoint fold {X Y : Type} (f : X -> Y -> Y) (l : list X) (b : Y) : Y
-
- Definition constfun {X : Type} (x : X) : nat -> X
- Definition ftrue := constfun true.
-
- Definition fold_length {X : Type} (l : list X) : nat := fold (fun _ n => S n) l 0.
- Theorem fold_length_correct : forall X (l : list X), fold_length l = length l.
- Definition fold_map {X Y : Type} (f : X -> Y) (l : list X) : (list Y) := fold (fun x t => f x :: t) l [].
- Theorem fold_map_correct : forall (X Y : Type) (f : X -> Y) (l : list X), fold_map f l = map f l.

Currying

- `Definition prod_curry {X Y Z : Type} (f : X * Y -> Z) (x : X) (y : Y) : Z := f (x, y).`
- `Definition prod_uncurry {X Y Z : Type} (f : X -> Y -> Z) (p : X * Y) : Z := f (fst p) (snd p).`
- `Theorem uncurry_curry : forall (X Y Z : Type) (f : X -> Y -> Z) x y, prod_curry (prod_uncurry f) x y = f x y.`
- `Theorem curry_uncurry: forall (X Y Z : Type) (f : X * Y -> Z) (p : X * Y), prod_uncurry (prod_curry f) p = f p.`

Church Numerals

- `Module Church`
-
- `End Church`

Problems

- `Theorem fold_length_correct` : power/mechanisms of `simpl.` vs. `reflexivity.`

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