Lists: Working with Structured Data (Lists.v)

Keywords

- `Search rev` : display a list of theorems involving `rev`
- `if ... then ... else ...` : conditional expression in `Coq` programming language

Syntax

- `Notation` The new `pair` notation can be used both in expressions and in pattern matches
- `multiple pattern` syntax

Tactics

- `rewrite -> A, B.` short for `rewrite -> A. rewrite -> B.`

Definitions and Theorems

- `Module NatList`
- `Inductive natprod : Type pair (n1 n2 : nat)`
- `Definition fst (p : natprod) : nat`
- `Definition snd (p : natprod) : nat`
- `Notation "( x , y )" := (pair x y).`
- `Definition swap_pair (p : natprod) : natprod`
- `Theorem surjective_pairing' : forall (n m : nat), (n, m) = (fst (n, m), snd (n, m)).`
- `Theorem surjective_pairing : forall (p : natprod), p = (fst p, snd p).`
- `Theorem snd_fst_is_swap : forall (p : natprod), (snd p, fst p) = swap_pair p.`

- Theorem fst_swap_is_snd : forall (p : natprod), fst (swap p) = snd p.
-
- Inductive natlist : Type : list of nat s
- Notation "x :: l" := (cons x l)
- Notation "[ ]" := nil.
- Notation "[x ; .. ; y] := (cons x .. (cons y nil) ..)."
-
- Fixpoint repeat (n count : nat) : natlist
- Fixpoint length (l : natlist) : nat
- Fixpoint app (l1 l2 : natlist) : natlist
- Notation "x ++ y" := (app x y)
-
- Definition hd (default : nat) (l : natlist) : nat
- Definition tl (l : natlist) : natlist
- Fixpoint nonzeros (l : natlist) : natlist
- Fixpoint oddmembers (l : natlist) : natlist
- Definition countoddmembers (l : natlist) : nat
- Fixpoint alternate (l1, l2 : natlist) : natlist
-
- Definition bag := natlist.
- Fixpoint count (v : nat) (s : bag) : nat
- Definition sum : bag -> bag -> bag := app.
- Definition add (v : nat) (s : bag) : bag := v :: s.
- Definition member (v : nat) (s : bag) : bool := 0 <? (count v s)
- Fixpoint remove_one (v : nat) (s : bag) : bag
- Fixpoint remove_all (v : nat) (s : bag) : bag
- Fixpoint subset (s1 : bag) (s2 : bag) : bool
-
- Theorem nil_app : forall l : natlist, [] ++ l = l.
- Theorem tl_length_pred : forall l : natlist, pred (length l) = length (tl l).
- Theorem app_assoc : forall l1 l2 l3 : natlist, (l1 ++ l2) ++ l3 = l1 ++ (l2 ++ l3).
- Fixpoint rev (l : natlist) : natlist
- Theorem app_length : forall l1 l2 : natlist, length (l1 ++ l2) = (length l1) + (length l2).
- Theorem rev_length : forall l : natlist, length (rev l) = length l.

-
- `Theorem app_nil_r : forall l : natlist, l ++ [] = l.`
- `Theorem rev_app_distr : forall l1 l2 : natlist, rev (l1 ++ l2) = rev l2 ++ rev l1.`
- `Theorem rev_involutive : forall l : natlist, rev (rev l) = l.`
- `Theorem app_assoc4 : forall l1 l2 l3 l4 : natlist, l1 ++ (l2 ++ (l3 ++ l4)) = ((l1 ++ l2) ++ l3) ++ l4.`
- `Theorem nonzeros_app : forall l1 l2 : natlist, nonzeros (l1 ++ l2) = nonzeros l1 + nonzeros l2.`
-
- `Fixpoint eqblist (l1 l2 : natlist) : bool`
- `Theorem eqblist_refl : forall l : natlist, true = eqblist l l.`
-
- `Theorem count_member_nonzero : forall (s : bag), 1 <=? (count 1 (1 :: s)) = true.`
- `Theorem leb_n_Sn : forall n : nat, n <=? (S n) = true.`
- `Theorem remove_does_not_increase_count : forall (s : bag), (count 0 (remove_one 0 s)) <=? (count 0 s) = true.`
-
- `Theorem rev_injective : forall (l1 l2 : natlist), rev l1 = rev l2 => l1 = l2.`
-
- `Fixpoint nth_bad (l : natlist) (n : nat) : nat`
- `Inductive natoption : Type`
- `Fixpoint nth_error (l : natlist) (n : nat) : natoption`
- `Definition option_elim (d : nat) (o : natoption) : nat`
- `Definition hd_error (l : natlist) : natoption`
- `Theorem option_elim_hd : forall (l : natlist) (default : nat), hd default l = option_elim default (hd_error l).`
-
- `End NatList.`

Partial Maps

- `Inductive id : Type`
- `Definition eqb_id (x1 x2 : id)`
- `Theorem eqb_id_refl : forall x : id, true = eqb_id x x.`

- `Module PartialMap`
- `Inductive partial_map : Type`
- `Defintion update (d : partial_map) (x : id) (value : nat) : partial_map`
- `Fixpoint find (x : id) (d : partial_map) : natoption`
- `Theorem update_eq : forall (d : partial_map) (x : id) (v : nat), find x (update d x v) = Some v.`
- `Theorem update_neq : forall (d : partial_map) (x y : id) (o : nat), eqb_id x y = false -> find x (update d y o) = find x d.`
- `End PartialMap`

Problems

- `Fixpoint count (v : nat) (s : bag) : nat` `v` in `v :: t` in `match s with` is not bound to `v : nat`?
- `Theorem app_assoc4 : forall l1 l2 l3 l4 : natlist, l1 ++ (l2 ++ (l3 ++ l4)) = ((l1 ++ l2) ++ l3) ++ l4.` It fails to use `rewrite -> app_assoc, app_assoc` to replace `rewrite -> app_assoc. rewrite -> app_assoc.` However, we can write `rewrite -> app_length, plus_comm.` in the proof of `Theorem rev_length`
- `Theorem nonzeros_app : forall l1 l2 : natlist, nonzeros (l1 ++ l2) = nonzeros l1 + nonzeros l2.`: simpler proof? without `destruct n`?

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