Logic Foundations

List: Working with Structured Data

熊英飞　胡振江

信息学院计算机系

2021年3月16日

北京大學 PEKING UNIVERSITY

Pairs of Numbers

A very **simple** structured data

Definition/Functions/Properties

Definition

```
Inductive natprod : Type :=
| pair (n1 n2 : nat).

Check (pair 3 5) : natprod.

Definition fst (p : natprod) : nat :=
  match p with
  | pair x y => x
  end.

Definition snd (p : natprod) : nat :=
  match p with
  | pair x y => y
  end.
```

Pair Patterns vs Multiple Patterns

```
Fixpoint minus (n m : nat) : nat :=
    match n, m with
    | O   , _    => O
    | S _ , O    => n
    | S n', S m' => minus n' m'
    end.

Definition bad_minus (n m : nat) : nat :=
    match n, m with
    | (O   , _  ) => O
    | (S _ , O  ) => n
    | (S n', S m') => bad_minus n' m'
    end.
```

Properties

```
Theorem surjective_pairing' : forall (n m : nat),
  (n,m) = (fst (n,m), snd (n,m)).
```

```
Proof.
  reflexivity. Qed.
```

```
Theorem surjective_pairing_stuck : ∀ (p : natprod),
p = (fst p, snd p).
```

```
Proof.
  intros p. destruct p as [n m]. simpl. reflexivity. Qed.
```

List of Numbers

A very **useful** structured data

Definition/Functions/Properties

List of Natural Numbers

A list is either the empty list or else a pair of a number and another list.

```
Inductive natlist : Type :=
 | nil
 | cons (n : nat) (l : natlist).

Definition mylist := cons 1 (cons 2 (cons 3 nil)).
```

```
Notation "x :: l" := (cons x l)
                   (at level 60, right associativity).
Notation "[ ]" := nil.
Notation "[ x ; .. ; y ]" := (cons x .. (cons y nil) ..).

Definition mylist1 := 1 :: (2 :: (3 :: nil)).
Definition mylist2 := 1 :: 2 :: 3 :: nil.
Definition mylist3 := [1;2;3].
```

Repeat

```
Fixpoint repeat (n count : nat) :
natlist :=
 match count with
 | O => nil
 | S count' => n :: (repeat n count')
 end.
```

Length

```
Fixpoint length (l:natlist) : nat :=
  match l with
  | nil => O
  | h :: t => S (length t)
  end.
```

Append

```
Fixpoint app (l1 l2 : natlist) : natlist :=
  match l1 with
  | nil    => l2
  | h :: t => h :: (app t l2)
  end.

Notation "x ++ y" := (app x y)
                     (right associativity, at level 60).

Example test_app1:        [1;2;3] ++ [4;5] = [1;2;3;4;5].
Proof. reflexivity. Qed.
Example test_app2:        nil ++ [4;5] = [4;5].
Proof. reflexivity. Qed.
```

Head & Tail

```
Definition hd (default : nat) (l : natlist) : nat :=
  match l with
  | nil => default
  | h :: t => h
  end.

Definition tl (l : natlist) : natlist :=
  match l with
  | nil => nil
  | h :: t => t
  end.

Example test_hd1:          hd 0 [1;2;3] = 1.
Proof. reflexivity. Qed.
Example test_hd2:          hd 0 [] = 0.
Proof. reflexivity. Qed.
Example test_tl:           tl [1;2;3] = [2;3].
Proof. reflexivity. Qed.
```

Bags via Lists

```
Definition bag := natlist.

Fixpoint count (v : nat) (s : bag) : nat
 (* REPLACE THIS LINE WITH ":= _your_definition_ ." *). Admitted.

Example test_count1:          count 1 [1;2;3;1;4;1] = 3.
 (* FILL IN HERE *) Admitted.
Example test_count2:          count 6 [1;2;3;1;4;1] = 0.
 (* FILL IN HERE *) Admitted.
```

Reasoning About Lists

```
Theorem nil_app : forall l : natlist,
  [] ++ l = l.
Proof. reflexivity. Qed.

Theorem tl_length_pred : forall l:natlist,
  pred (length l) = length (tl l).
Proof.
  intros l. destruct l as [| n l'].
  - (* l = nil *)
    reflexivity.
  - (* l = cons n l' *)
    reflexivity.  Qed.
```

Induction on Lists

```
Theorem app_assoc : forall l1 l2 l3 : natlist,
 (l1 ++ l2) ++ l3 = l1 ++ (l2 ++ l3).
Proof.
 intros l1 l2 l3. induction l1 as [| n l1' IHl1'].
 - (* l1 = nil *)
  reflexivity.
 - (* l1 = cons n l1' *)
  simpl. rewrite -> IHl1'. reflexivity.  Qed.
```

Reversing a List: Using Auxiliary Lemma

```
Fixpoint rev (l:natlist) : natlist :=
 match l with
 | nil    => nil
 | h :: t => rev t ++ [h]
 end.

Theorem rev_length_firsttry : forall l : natlist,
 length (rev l) = length l.
Proof.
 intros l. induction l as [| n l' IHl'].
 - (* l = nil *)
  reflexivity.
 - (* l = n :: l' *)
  simpl.
Abort.
```


北京大學 PEKING UNIVERSITY

Reversing a List: Using Auxiliary Lemma

```
Lemma app_length : forall l1 l2 : natlist,
 length (l1 ++ l2) = (length l1) + (length l2).
Proof.
intros l1 l2. induction l1 as [| n l1' IHl1'].
 - (* l1 = nil *)
   reflexivity.
 - (* l1 = cons *)
   simpl. rewrite -> IHl1'. reflexivity.  Qed.

Theorem rev_length_firsttry : forall l : natlist,
 length (rev l) = length l.
Proof.
 intros l. induction l as [| n l' IHl'].
 - (* l = nil *)
   reflexivity.
 - (* l = cons *)
   simpl. rewrite -> app_length.
   simpl. rewrite -> IHl'. rewrite plus_comm.
   reflexivity.
Qed.
```

Search Properties

- Search rev.
  - display a list of all theorems involving rev
- Search (_ + _ = _ + _).
  - search for all theorems involving the equality of two additions
- Search (_ + _ = _ + _) inside Induction.
  - search inside a particular module
- Search (?x + ?y = ?y + ?x).
  - using variables in the search pattern instead of wildcards

Option

Dealing with Exception

Dealing with Exception

```
Inductive natoption : Type :=
 | Some (n : nat)
 | None.

Fixpoint nth_error (l:natlist) (n:nat) : natoption :=
 match l with
 | nil => None
 | a :: l' => match n with
            | O => Some a
            | S n' => nth_error l' n'
            end
 end.

Fixpoint nth_error' (l:natlist) (n:nat) : natoption :=
 match l with
 | nil => None
 | a :: l' => if n =? O then Some a else nth_error' l' (pred n)
 end.
```

Partial Map

Key-value Correspondence

Definition

```
Inductive partial_map : Type :=
  | empty
  | record (i : id) (v : nat) (m : partial_map).

Inductive id : Type :=
  | Id (n : nat).

Definition eqb_id (x1 x2 : id) :=
  match x1, x2 with
  | Id n1, Id n2 => n1 =? n2
  end.
```


北京大學 PEKING UNIVERSITY

Functions

```
Definition update (d : partial_map)
               (x : id) (value : nat)
               : partial_map :=
  record x value d.

Fixpoint find (x : id) (d : partial_map) : natoption :=
  match d with
  | empty        => None
  | record y v d' => if eqb_id x y
                     then Some v
                     else find x d'
  end.
```

作业

- 完成Lists.v中的至少10个练习题。