Logic Foundations
Polymorphism and Higher-Order Functions

熊英飞　胡振江

信息学院计算机系

2021年3月19日

Polymorphic Lists

Definition

```
Inductive list (X:Type) : Type :=
 | nil
 | cons (x : X) (l : list X).

Check list : Type -> Type.

Check nil : forall X : Type, list X.
Check cons : forall X : Type, X -> list X -> list X.

Check (nil nat) : list nat.

Check (cons nat 3 (nil nat)) : list nat.
```

Functions

```
Fixpoint repeat (X : Type) (x : X) (count : nat) : list X :=
  match count with
  | 0 => nil X
  | S count' => cons X x (repeat X x count')
  end.

Example test_repeat1 :
  repeat nat 4 2 = cons nat 4 (cons nat 4 (nil nat)).
Proof. reflexivity. Qed.

Example test_repeat2 :
  repeat bool false 1 = cons bool false (nil bool).
Proof. reflexivity. Qed.
```

Type Annotation Inference

```
Fixpoint repeat' X x count : list X :=
 match count with
 | o      => nil X
 | S count' => cons X x (repeat' X x count')
 end.

Check repeat'
 : forall X : Type, X -> nat -> list X.
Check repeat
 : forall X : Type, X -> nat -> list X.
```

This powerful type annotation inference facility means we don't always have to write explicit type annotations everywhere.

Implicit Arguments

- Arguments Directive

```
Arguments nil {X}.
Arguments cons {X} _ _.
Arguments repeat {X} x count.

Definition list123'' := cons 1 (cons 2 (cons 3 nil)).
```

Implicit Arguments

- Declare an argument to be implicit in definition functions.

```
Fixpoint repeat''' {X : Type} (x : X) (count : nat) : list X :=
  match count with
  | o        => nil
  | S count' => cons x (repeat''' x count')
  end.
```

We don't do on types:

```
Inductive list' {X:Type} : Type :=
  | nil'
  | cons' (x : X) (l : list').
```

Implicit Arguments

- Declare an argument to be implicit in definition functions.

```
Fixpoint rev {X:Type} (l:list X) : list X :=
  match l with
  | nil      => nil
  | cons h t => app (rev t) (cons h nil)
  end.

Example test_rev1 :
  rev (cons 1 (cons 2 nil)) = (cons 2 (cons 1 nil)).
Proof. reflexivity. Qed.
Example test_rev2:
  rev (cons true nil) = cons true nil.
Proof. reflexivity. Qed.
```

Suppling Type Arguments Explicitly

Suppose we write

```
Fail Definition mynil := nil.
```

Coq gives us an error because it does not know that type augment to supply to nil. We should supply it by

```
Definition mynil : list nat := nil.
```

or by

```
Definition mynil' := @nil nat.
```

Polymorphic Pairs

Definition

```
Inductive prod (X Y : Type) : Type :=
| pair (x : X) (y : Y).

Arguments pair {X} {Y} _ _.

Notation "( x , y )" := (pair x y).
Notation "X * Y" := (prod X Y) : type_scope.
```

Functions

```
Definition fst {X Y : Type} (p : X * Y) : X :=
  match p with
  | (x, y) => x
  end.

Definition snd {X Y : Type} (p : X * Y) : Y :=
  match p with
  | (x, y) => y
  end.

Fixpoint combine {X Y : Type} (lx : list X) (ly : list Y)
           : list (X*Y) :=
  match lx, ly with
  | [], _ => []
  | _, [] => []
  | x :: tx, y :: ty => (x, y) :: (combine tx ty)
  end.
```

Polymorphic Option

Definition

```
Inductive option (X:Type) : Type :=
  | Some (x : X)
  | None.

Arguments Some {X} _.
Arguments None {X}.
```

Functions

```
Fixpoint nth_error {X : Type} (l : list X) (n : nat)
              : option X :=
 match l with
 | nil => None
 | a :: l' => match n with
          | O => Some a
          | S n' => nth_error l' n'
          end
 end.

Example test_nth_error1 : nth_error [4;5;6;7] 0 = Some 4.
Proof. reflexivity. Qed.
Example test_nth_error2 : nth_error [[1];[2]] 1 = Some [2].
Proof. reflexivity. Qed.
Example test_nth_error3 : nth_error [true] 2 = None.
Proof. reflexivity. Qed.
```

Functions as Data

Higher-Order Functions

```
Definition doit3times {X:Type} (f:X->X) (n:X) : X :=
  f (f (f n)).

Check @doit3times : forall X : Type, (X -> X) -> X -> X.

Example test_doit3times: doit3times minustwo 9 = 3.
Proof. reflexivity. Qed.

Example test_doit3times': doit3times negb true = false.
Proof. reflexivity. Qed.
```

Filters

```
Fixpoint filter {X:Type} (test: X->bool) (l:list X) : (list X) :=
  match l with
  | [] => []
  | h :: t =>
    if test h then h :: (filter test t)
    else filter test t
  end.

Example test_filter1: filter evenb [1;2;3;4] = [2;4].
Proof. reflexivity. Qed.

Definition length_is_1 {X : Type} (l : list X) : bool :=
  (length l) =? 1.
Example test_filter2:
    filter length_is_1
           [ [1; 2]; [3]; [4]; [5;6;7]; []; [8] ]
  = [ [3]; [4]; [8] ].
Proof. reflexivity. Qed.
```

Anonymous Functions

```
Example test_anon_fun':
  doit3times (fun n => n * n) 2 = 256.
Proof. reflexivity. Qed

Example test_filter2':
   filter (fun l => (length l) =? 1)
          [ [1; 2]; [3]; [4]; [5;6;7]; []; [8] ]
  = [ [3]; [4]; [8] ].
Proof. reflexivity. Qed.
```

Map

```
Fixpoint map {X Y: Type} (f:X->Y) (l:list X) : (list Y) :=
 match l with
 | []     => []
 | h :: t => (f h) :: (map f t)
 end.

Example test_map1: map (fun x => plus 3 x) [2;0;2] = [5;3;5].
Proof. reflexivity. Qed.

Example test_map2:
 map oddb [2;1;2;5] = [false;true;false;true].
Proof. reflexivity. Qed.

Example test_map3:
   map (fun n => [evenb n;oddb n]) [2;1;2;5]
 = [[true;false];[false;true];[true;false];[false;true]].
Proof. reflexivity. Qed.
```

Fold

```
Fixpoint fold {X Y: Type} (f: X->Y->Y) (l: list X) (b: Y) : Y :=
  match l with
  | nil => b
  | h :: t => f h (fold f t b)
  end.

Check (fold andb) : list bool -> bool -> bool.

Example fold_example1 :
  fold mult [1;2;3;4] 1 = 24.
Proof. reflexivity. Qed.

Example fold_example2 :
  fold andb [true;true;false;true] true = false.
Proof. reflexivity. Qed.

Example fold_example3 :
  fold app  [[1];[];[2;3];[4]] [] = [1;2;3;4].
Proof. reflexivity. Qed.
```

Functions that Construct Functions

```
Definition constfun {X: Type} (x: X) : nat->X :=
  fun (k:nat) => x.

Example constfun_example2 : (constfun 5) 99 = 5.
Proof. reflexivity. Qed.

Definition plus3 := plus 3.
Check plus3 : nat -> nat.

Example test_plus3 :    plus3 4 = 7.
Proof. reflexivity. Qed.
Example test_plus3' :   doit3times plus3 0 = 9.
Proof. reflexivity. Qed.
```

作业

- 完成Poly.v中的至少10个练习题。