函数语言程序设计10.29

10205101530-赵晗瑜

- 定义函数max使得(max L)返回类型为natoption：当自然数列表L为空时返回None,否责返回Some n，其中n为L中最大元素。

  [ans]

```
Fixpoint max(L:natlist):natoption:=
  match L with
   |nil=>None
   |n::L'=>match max L' with
           |None=>Some n
           |Some m =>if ( m<=?n) then Some n
                     else Some m
           end
end.
Example test_max1 :max [1;2;3] = Some 3.
Proof. reflexivity.  Qed.
Example test_max2 :max [4;2;3] = Some 4.
Proof. reflexivity.  Qed.
Example test_max4 :max [4;2;5] = Some 5.
Proof. reflexivity.  Qed.
Example test_max3 :max [] = None.
Proof. reflexivity. Qed.
```

  [运行结果]

```
Fixpoint max(L:natlist):natoption:=
  match L with
   |nil=>None
   |n::L'=>match max L' with
           |None=>Some n
           |Some m =>if ( m<=?n) then Some n
                     else Some m
           end
end.
Example test_max1 :max [1;2;3] = Some 3.
Proof. reflexivity.  Qed.
Example test_max2 :max [4;2;3] = Some 4.
Proof. reflexivity.  Qed.
Example test_max4 :max [4;2;5] = Some 5.
Proof. reflexivity.  Qed.
Example test_max3 :max [] = None.
Proof. reflexivity. Qed.
```

- **练习 2.29.** 定义函数 maxPair，把输入的一个自然数列表中最大的奇数和偶数找出来，组成一个二元组作为返回值。如果列表中没有奇数或偶数，则用 0 替代。例如，

[ans]

```
Fixpoint maxPair(l : natlist) : natprod :=
  match l with
   |nil=>(0,0)
   |a::l'=>match maxPair l' with
           |(n,m)=>if even a && (m<=?a) then (n,a)
                    else if odd a && (n<=?a) then (a,m)
                    else (n,m)
           end
  end.


Example test_maxPair1: maxPair [1;2;5;4;8;10;3] = (5, 10).
Proof. reflexivity. Qed.

Example test_maxPair2: maxPair [2;4] = (0, 4).
Proof. reflexivity. Qed.
```

[运行结果]

```
Fixpoint maxPair(l : natlist) : natprod :=
  match l with
   |nil=>(0,0)
   |a::l'=>match maxPair l' with
           |(n,m)=>if even a && (m<=?a) then (n,a)
                   else if odd a && (n<=?a) then (a,m)
                   else (n,m)
           end
  end.

Example test_maxPair1: maxPair [1;2;5;4;8;10;3] = (5, 10).
Proof. reflexivity. Qed.

Example test_maxPair2: maxPair [2;4] = (0, 4).
Proof. reflexivity. Qed.
```

- 3.

```
Theorem rev_app_distr: ∀ X (l1 l2 : list X),
    rev (l1 ++ l2) = rev l2 ++ rev l1.
Proof.
    (* FILL IN HERE *) Admitted.
```

[ans]

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

[运行结果]

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

- 4.

```
Theorem rev_involutive : ∀ X : Type, ∀ l : list X,
    rev (rev l) = l.
Proof.
    (* FILL IN HERE *) Admitted.
```

[ans]

```
Theorem rev_involutive : forall l : natlist,
  rev (rev l) = l.
Proof.
  intros l. induction l as [|n l' IHl'].
  - reflexivity.
  - simpl. rewrite->rev_app_distr. simpl. rewrite->IHl'.
reflexivity.
Qed.
```

[运行结果]

```
(** An _involution_ is a function that is its own inverse. That is,
    applying the function twice yield the original input. *)
Theorem rev_involutive : forall l : natlist,
  rev (rev l) = l.
Proof.
  intros l. induction l as [|n l' IHl'].
  - reflexivity.
  - simpl. rewrite->rev_app_distr. simpl. rewrite->IHl'. reflexivity.
Qed.
```