函数语言作业10

10205101530-赵晗瑜

Theorem dist_exists_or : ∀ (X:Type) (P Q : X → Prop),
(∃ x, P x ∨ Q x) ↔ (∃ x, P x) ∨ (∃ x, Q x).
Proof.
(* FILL IN HERE *) Admitted.

【ans】

```
Theorem dist_exists_or : forall (X:Type) (P Q : X -> Prop),
  (exists x, P x \/ Q x) <-> (exists x, P x) \/ (exists x, Q x).
Proof.
   (* FILL IN HERE *)
  intros X P Q. split.
  - intros [x [H | H]].
    + left. exists x. apply H.
    + right. exists x. apply H.
  - intros [[x Px] | [x Qx]].
    + exists x. left. apply Px.
    + exists x. right. apply Qx.
Qed.
```

【运行截图】


```
Theorem dist_exists_or : forall (X:Type) (P Q : X -> Prop),
  (exists x, P x \/ Q x) <-> (exists x, P x) \/ (exists x, Q x).
Proof.
   (* FILL IN HERE *)
  intros X P Q. split.
  - intros [x [H | H]].
    + left. exists x. apply H.
    + right. exists x. apply H.
  - intros [[x Px] | [x Qx]].
    + exists x. left. apply Px.
    + exists x. right. apply Qx.
Qed.
```

Inductively define a relation CE such that (CE m n) holds iff m and n are two consecutive even numbers with m smaller than n.

Example test_CE : CE 4 6.

Proof. (* Fill in here *) Admitted.

【ans】

```
Inductive CE : nat -> nat -> Prop :=
  | CE_base: CE 0 2
  | CE_inductive (n m : nat) (H : CE n m): CE (S (S n)) (S (S m)).
Example test_CE: CE 4 6.
Proof.
  apply CE_inductive. apply CE_inductive. apply CE_base.
Qed.
```

【运行截图】

```
Inductive CE : nat -> nat -> Prop :=
  | CE_base: CE 0 2
  | CE_inductive (n m : nat) (H : CE n m): CE (S (S n)) (S (S m)).
Example test_CE: CE 4 6.
Proof.
  apply CE_inductive. apply CE_inductive. apply CE_base.
Qed.
```

Theorem CE_SS : forall n m, CE (S (S n)) (S (S m)) -> CE n m.

Proof. (* Fill in here *) Admitted.

【ans】

```
Lemma CE_inversion: forall (n m : nat),
  CE n m -> (n = 0 /\ m = 2) \/ (exists n' m', n = S (S n') /\ m =
S (S m') /\ (CE n' m')).
Proof.
  intros n m H.
  destruct H as [| n' m' H'].
  - left. split.
    + reflexivity.
    + reflexivity.
  - right.
    exists n'. exists m'.
```

```
    split.
    + reflexivity.
    + split.
      * reflexivity.
      * apply H'.
Qed.

Theorem CE_SS: forall n m,
  CE (S (S n)) (S (S m)) -> CE n m.
Proof.
  intros n m H.
  apply CE_inversion in H.
  destruct H as [H | H].
  - destruct H as [H1 H2].
    discriminate H1.
  - destruct H as [n'].
    destruct H as [m'].
    destruct H as [Hn [Hm HCE]].
    injection Hn as Hn.
    injection Hm as Hm.
    rewrite Hn.
    rewrite Hm.
    apply HCE.
Qed.
```

【运行截图】

```
Lemma CE_inversion: forall (n m : nat),
  CE n m -> (n = 0 /\ m = 2) \/ (exists n' m', n = S (S n') /\ m = S (S m') /\ (CE n' m')).
Proof.
  intros n m H.
  destruct H as [| n' m' H'].
  - left. split.
    + reflexivity.
    + reflexivity.
  - right.
    exists n'. exists m'.
    split.
    + reflexivity.
    + split.
      * reflexivity.
      * apply H'.
Qed.

Theorem CE_SS: forall n m,
  CE (S (S n)) (S (S m)) -> CE n m.
Proof.
  intros n m H.
  apply CE_inversion in H.
  destruct H as [H | H].
  - destruct H as [H1 H2].
    discriminate H1.
  - destruct H as [n'].
    destruct H as [m'].
    destruct H as [Hn [Hm HCE]].
    injection Hn as Hn.
    injection Hm as Hm.
    rewrite Hn.
    rewrite Hm.
    apply HCE.
Qed.
```

Theorem In_app_iff : ∀ A l l' (a:A),
 In a (l++l') ↔ In a l ∨ In a l'.
Proof.
 intros A l. induction l as [|a' l' IH].
 (* FILL IN HERE *) Admitted.

【ans】

```
Theorem In_app_iff : forall A l l' (a:A),
  In a (l++l') <-> In a l \/ In a l'.
Proof.
  (* FILL IN HERE *)
  intros A l l' a.
  induction l as [|h t IHt].
  - simpl. split.
    + intro H. right; apply H.
    + intros [[]|H]. apply H.
  - simpl. split.
    + intros [H|H].
      * left; left; apply H.
      * apply IHt in H. destruct H as [H1|H2].
        { left; right; apply H1. }
```

```
          { right; apply H2. }
      + intros [[H|H]|H].
        * left; apply H.
        * right. rewrite IHt; left; apply H.
        * right; rewrite IHt; right; apply H.
Qed.
```

【运行截图】

```
(** **** Exercise: 2 stars, standard (In_app_iff) *)
Theorem In_app_iff : forall A l l' (a:A),
  In a (l++l') <-> In a l \/ In a l'.
Proof.
  (* FILL IN HERE *)
  intros A l l' a.
  induction l as [|h t IHt].
  - simpl. split.
    + intro H. right; apply H.
    + intros [[]|H]. apply H.
  - simpl. split.
    + intros [H|H].
      * left; left; apply H.
      * apply IHt in H. destruct H as [H1|H2].
        { left; right; apply H1. }
        { right; apply H2. }
    + intros [[H|H]|H].
      * left; apply H.
      * right. rewrite IHt; left; apply H.
      * right; rewrite IHt; right; apply H.
Qed.
```

Drawing inspiration from In, write a recursive function All stating that some property P holds of all elements of a list l. To make sure your definition is correct, prove the All_In lemma below. (Of course, your definition should *not*just restate the left-hand side of All_In.)

Fixpoint All {T : Type} (P : T → Prop) (l : list T) : Prop
(* REPLACE THIS LINE WITH ":= *your_definition* ." *). Admitted.

Theorem All_In :
∀ T (P : T → Prop) (l : list T),
(∀ x, In x l → P x) ↔
All P l.
Proof.
(* FILL IN HERE *) Admitted.

【ans】

```
Fixpoint All {T : Type} (P : T -> Prop) (l : list T) : Prop
  (* REPLACE THIS LINE WITH ":= _your_definition_ ." *)
:=  match l with
  | [] => True
  | h :: t => P h /\ All P t
  end.

Theorem All_In :
  forall T (P : T -> Prop) (l : list T),
    (forall x, In x l -> P x) <->
    All P l.
Proof.
  (* FILL IN HERE *)
  intros T P l. induction l as [|x' l' IHl'].
  - simpl. split.
    + intros; apply I.
    + intros _ x F. inversion F.
  - simpl. split.
    + intro H. split.
      * apply H. left; reflexivity.
      * apply IHl'. intros x Hin. apply H. right; apply Hin.
    + intros [H1 H2]. intros x [H|H].
      * rewrite <- H; apply H1.
      * rewrite <- IHl' in H2. apply H2. apply H.
Qed.
```

【运行截图】

```
Fixpoint All {T : Type} (P : T -> Prop) (l : list T) : Prop
  (* REPLACE THIS LINE WITH ":= _your_definition_ ." *)
:=  match l with
  | [] => True
  | h :: t => P h /\ All P t
  end.

Theorem All_In :
  forall T (P : T -> Prop) (l : list T),
    (forall x, In x l -> P x) <->
    All P l.
Proof.
  (* FILL IN HERE *)
  intros T P l. induction l as [|x' l' IHl'].
  - simpl. split.
    + intros; apply I.
    + intros _ x F. inversion F.
  - simpl. split.
    + intro H. split.
      * apply H. left; reflexivity.
      * apply IHl'. intros x Hin. apply H. right; apply Hin.
    + intros [H1 H2]. intros x [H|H].
      * rewrite <- H; apply H1.
      * rewrite <- IHl' in H2. apply H2. apply H.
Qed.
```