Chapter 1

Library c07_indprop

Capítulo 7 - Inductively Defined Propositions (IndProp)

`From` *LF* `Require` `Export` *c06_logic*.

`Require` *Coq.omega.Omega*.

`Inductive` *ev* : *nat* → `Prop` :=
| *ev_0* : *ev* 0
| *ev_SS* : ∀ *n* : *nat*, *ev n* → *ev* (*S* (*S n*)).

`Theorem` *ev_4* : *ev* 4.

`Theorem` *ev_4'* : *ev* 4.

`Theorem` *ev_plus4* : ∀ *n*, *ev n* → *ev* (4 + *n*).

Exercise: 1 star, standard (ev_double)

`Theorem` *ev_double* : ∀ *n*, *ev* (*double n*).

`Theorem` *ev_minus2* : ∀ *n*, *ev n* → *ev* (*pred* (*pred n*)).

`Theorem` *ev_minus2'* : ∀ *n*, *ev n* → *ev* (*pred* (*pred n*)).

`Theorem` *evSS_ev* : ∀ *n*, *ev* (*S* (*S n*)) → *ev n*.

`Theorem` *one_not_even* : ¬ *ev* 1.

Exercise: 1 star (SSSSev__even) `Theorem` *SSSSev__even* : ∀ *n*, *ev* (*S* (*S* (*S* (*S n*)))) → *ev n*.

Exercise: 1 star (even5_nonsense) `Theorem` *even5_nonsense* : *ev* 5 → 2 + 2 = 9.

Falta fazer

Lemma ev_even : forall n, ev n -> exists k, n = double k. Proof. intros n E. induction E as |*n' E' IH*.

- exists 0. reflexivity.
- destruct IH as *k' Hk'*. rewrite Hk'. exists (S k'). reflexivity.

Qed.

Theorem ev_even_iff : forall n, ev n <-> exists k, n = double k. Proof. intros n. split.

- apply ev_even.
- intros *k Hk*. rewrite Hk. apply ev_double.

Qed.

Exercise: 2 stars (ev_sum) `Theorem` *ev_sum* : $\forall$ *n m*, *ev n* $\rightarrow$ *ev m* $\rightarrow$ *ev* (*n* + *m*).

Exercise: 4 stars, advanced, optional (ev'_ev) `Inductive` *ev'* : *nat* $\rightarrow$ `Prop` :=
| *ev'_0* : *ev'* 0
| *ev'_2* : *ev'* 2
| *ev'_sum* : $\forall$ *n m*, *ev' n* $\rightarrow$ *ev' m* $\rightarrow$ *ev'* (*n* + *m*).

`Theorem` *ev'_ev* : $\forall$ *n*, *ev' n* $\leftrightarrow$ *ev n*.

Exercise: 3 stars, advanced, recommended (ev_ev__ev) `Theorem` *ev_ev__ev* : $\forall$ *n m*, *ev* (*n*+*m*) $\rightarrow$ *ev n* $\rightarrow$ *ev m*.

Exercise: 3 stars, optional (ev_plus_plus) `Theorem` *ev_plus_plus* : $\forall$ *n m p*, *ev* (*n*+*m*) $\rightarrow$ *ev* (*n*+*p*) $\rightarrow$ *ev* (*m*+*p*).

`Inductive` *le* : *nat* $\rightarrow$ *nat* $\rightarrow$ `Prop` :=
| *le_n* : $\forall$ *n*, *le n n*
| *le_S* : $\forall$ *n m*, (*le n m*) $\rightarrow$ (*le n* (*S m*)).

`Notation` "m <= n" := (*le m n*).

`Theorem` *test_le1* : $3 \leq 3$.

`Theorem` *test_le2* : $3 \leq 6$.

`Theorem` *test_le3* : $(2 \leq 1) \rightarrow 2 + 2 = 5$.

`Definition` *lt* (*n m*:*nat*) := *le* (*S n*) *m*.

`Notation` "m < n" := (*lt m n*).

`Inductive` *square_of* : *nat* $\rightarrow$ *nat* $\rightarrow$ `Prop` :=
| *sq* : $\forall$ *n*:*nat*, *square_of n* ($n \times n$).

`Inductive` *next_nat* : *nat* $\rightarrow$ *nat* $\rightarrow$ `Prop` :=
| *nn* : $\forall$ *n*:*nat*, *next_nat n* (*S n*).

`Inductive` *next_even* : *nat* $\rightarrow$ *nat* $\rightarrow$ `Prop` :=
| *ne_1* : $\forall$ *n*, *ev* (*S n*) $\rightarrow$ *next_even n* (*S n*)
| *ne_2* : $\forall$ *n*, *ev* (*S* (*S n*)) $\rightarrow$ *next_even n* (*S* (*S n*)).

Exercise: 2 stars, optional (total_relation)

Lemma le_trans : forall m n o, m <= n -> n <= o -> m <= o. Proof. intros m n o Hmn Hno. rewrite <-Hno. Qed.

Theorem O_le_n : forall n, 0 <= n. Proof. induction n as |*n' IHn*.

- apply le_n.
- apply le_S. apply IHn.

Qed.

Theorem n_le_m__Sn_le_Sm : forall n m, n <= m -> S n <= S m. Proof. intros n m H. induction H.

- apply le_n.
- apply le_S. apply IHle.

Qed.

Theorem Sn_le_Sm__n_le_m : forall n m, S n <= S m -> n <= m. Proof. intros n m H. inversion H.

- apply le_n.
- generalize dependent H1. apply le_trans. apply le_S. apply le_n.

Qed.

Theorem le_plus_l : forall a b, a <= a + b. Proof. intros a b. induction a.

- apply O_le_n.
- apply n_le_m__Sn_le_Sm in IHa. apply IHa.

Qed.

Theorem plus_lt : forall n1 n2 m, n1 + n2 < m -> n1 < m /\ n2 < m. Proof. unfold lt. intros n1 n2 m. split.

- apply (le_trans (S n1) (S (n1 + n2))). + apply n_le_m__Sn_le_Sm. apply le_plus_l. + apply H.
- apply (le_trans (S n2) (S (n1 + n2))). + apply n_le_m__Sn_le_Sm. rewrite -> plus_comm. apply le_plus_l. + apply H.

Qed.

Theorem lt_S : forall n m, n < m -> n < S m. Proof. unfold lt. intros n m H. apply le_S. apply H. Qed.

Theorem leb_complete : forall n m, leb n m = true -> n <= m. Proof. intros n m. generalize dependent n. induction m as |*m' IHm*.

- intros n H. destruct n as |*n'*. + apply le_n. + inversion H.
- intros n H. destruct n as |*n'*. + apply O_le_n. + simpl in H. apply IHm in H. apply n_le_m__Sn_le_Sm. apply H.

Qed.

Theorem leb_correct : forall n m, n <= m -> leb n m = true. Proof. intros n m. generalize dependent n. induction m as |*m' IHm*.

- intros n H. inversion H. reflexivity.

- intros n H. destruct n. + reflexivity. + apply IHm. generalize dependent H. apply Sn_le_Sm__n_le_m .

Qed.

Theorem leb_true_trans : forall n m o, leb n m = true -> leb m o = true -> leb n o = true. Proof. intros n m o H H0. apply leb_complete in H. apply leb_complete in H0. apply leb_correct. generalize dependent H0. generalize dependent H. apply le_trans. Qed.

Theorem leb_iff : forall n m, leb n m = true <-> n <= m. Proof. split.

- apply leb_complete.

- apply leb_correct.

Qed.