

Tipos indutivos e indução

Introdução ao assistente de provas Coq

Rodrigo Ribeiro

Tipos indutivos

- ▶ Em Coq tudo é definido usando tipos indutivos, exceto tipos de funções e quantificador universal.

```
Inductive bool : Set :=  
| true : bool | false : bool
```

Tipos indutivos

- ▶ Definição de funções por *pattern matching*.

```
Definition not_bool (b : bool) : bool :=  
  match b with  
  | false => true  
  | true  => false  
end.
```

- ▶ Alternativas devem cobrir todas as possibilidades.
 - ▶ Todas as funções deve ser totais!

Tipos indutivos

- Uma demonstração...

```
Lemma not_bool_inv : forall b : bool,  
    not_bool (not_bool b) = b.
```

```
Proof.
```

```
  intro b.
```

```
  destruct b.
```

```
  +
```

```
    simpl.
```

```
    reflexivity.
```

```
  +
```

```
    simpl.
```

```
    reflexivity.
```

```
Qed.
```

Tipos indutivos

- ▶ Números naturais

```
Inductive nat : Set :=  
| 0 : nat  
| S : nat -> nat.
```

Tipos indutivos

- ▶ Adição de números naturais
 - ▶ Funções recursivas são definidas usando `Fixpoint`.
 - ▶ Funções devem usar recursão estrutural.

```
Fixpoint add (n m : nat) : nat :=  
  match n with  
  | 0      => m  
  | S n' => S (add n' m)  
  end.
```

Tipos indutivos

- Uma propriedade simples...

Lemma zero_identity_add_left : forall n, 0 + n = n.

Proof.

intro n.

simpl.

reflexivity.

Qed.

Tipos indutivos

- ▶ Outra propriedade simples...

Lemma zero_identity_add_right : forall n, $n + 0 = n$.

Proof.

intro n.

simpl.

reflexivity.

Tipos indutivos

- ▶ Mas executar `reflexivity` resulta em...

```
Error: In environment
```

```
n : nat
```

```
Unable to unify n with n + 0.
```

Tipos indutivos

- ▶ Problema: igualdade em teoria de tipos.
 - ▶ Várias definições, problema de pesquisa em aberto.
 - ▶ Diferença entre $n + 0$ e $0 + n$.
- ▶ Igualdade proposicional.

```
Inductive eq (A : Type) (x : A) : A -> Prop :=  
| eq_refl : eq A x
```

Tipos indutivos

- Tentando com a tática destruct

Lemma zero_identity_add_right : forall n, $n + 0 = n$.

Proof.

```
  intro n.
```

```
  destruct n.
```

```
  +
```

```
      simpl.
```

```
      reflexivity.
```

```
  +
```

```
      simpl.
```

```
      reflexivity.
```

Qed.

Tipos indutivos

- ▶ Caso $n = 0$ ok!
- ▶ Caso $n = S\ n'$:

Error: In environment

$n : \text{nat}$

Unable to unify n with $n + 0$.

Tipos indutivos

- ▶ O que precisamos é de indução matemática!
 - ▶ Para isso, usaremos a tática `induction`.

Indução em Coq

Lemma zero_identity_add_right : forall n, n + 0 = n.

Proof.

intro n.

induction n as [| n' IHn'].

+

simpl.

reflexivity.

+

simpl.

rewrite IHn'.

reflexivity.

Qed.

Indução em Coq

Lemma add_inc : forall m n, S (m + n) = m + S n.

Proof.

intros m n.

induction m as [| m' IHm'].

+

simpl.

reflexivity.

+

simpl.

rewrite IHm'.

reflexivity.

Qed.

Indução em Coq

Lemma add_commut : forall n m, $n + m = m + n$.

Proof.

 intros n m.

 induction n as [| n' IHn'].

 +

 simpl.

 symmetry.

 apply zero_identity_add_right.

 +

 simpl.

 rewrite IHn'.

 apply add_inc.

Qed.

Listas em Coq

► Listas

```
Inductive list (A : Type) : Type :=  
  | nil : list A  
  | cons : A -> list A -> list A.
```

Funções sobre Listas

```
Fixpoint repeat {A : Type}(n : nat)(x : A) : list A :=  
  match n with  
  | 0 => []  
  | S n' => x :: repeat n' x  
  end.
```

```
Fixpoint length {A : Type}(xs : list A) : nat :=  
  match xs with  
  | [] => 0  
  | (x :: xs) => S (length xs)  
  end.
```

Concatenação de listas

```
Fixpoint app {A : Type}(xs ys : list A) : list A :=  
  match xs with  
  | []          => ys  
  | (x :: xs) => x :: (app xs ys)  
end.
```

Indução sobre listas

```
Lemma length_app
: forall {A : Type}(xs ys : list A),
    length (xs ++ ys) = length xs + length ys.
```

Proof.

```
  intros A xs ys.
  induction xs as [| z zs IHzs].
  +
    simpl.
    reflexivity.
  +
    simpl.
    rewrite IHzs.
    reflexivity.
```

Qed.

Predicados indutivos

```
Inductive even : nat -> Prop :=  
| ev_zero : even 0  
| ev_ss    : forall n, even n -> even (S (S n)).
```

Usando o predicado even

Example `eight_is_even` : even 8.

Proof.

 apply ev_ss.

 apply ev_ss.

 apply ev_ss.

 apply ev_ss.

 apply ev_zero.

Qed.

Princípio de inversão

- ▶ Deduzir premissas a partir de uma certa proposição.
 - ▶ Disponível pela tática `inversion`.
- ▶ Proposições não demonstráveis e inversão.

```
Lemma one_not_even : ~ even 1.
```

```
Proof.
```

```
  intros Hone.
```

```
  inversion Hone.
```

```
Qed.
```

Princípio de inversão

Lemma even_2_inv

: forall n, even (2 + n) -> even n.

Proof.

intros n H.

inversion H.

assumption.

Qed.

Relação de ordem sobre naturais

```
Inductive le (n : nat) : nat -> Prop :=  
| le_n : n <= n  
| le_S : forall m : nat, n <= m -> n <= S m
```

Relação de ordem sobre naturais

Example teste_le : 3 <= 6.

Proof.

 apply le_S.

 apply le_S.

 apply le_S.

 apply le_n.

Qed.

Relação de ordem sobre naturais

Lemma le_0_n : forall n, 0 <= n.

Proof.

intros n.

induction n as [| n' IHn'].

+

apply le_n.

+

apply le_S.

assumption.

Qed.

Relação de ordem sobre naturais

Lemma le_cong_S : forall n m, n <= m -> S n <= S m.

Proof.

 intros n m Hnm.

 induction Hnm.

 +

 apply le_refl.

 +

 apply le_S.

 assumption.

Qed.