Inductive data types (II)

jean-jacques.levy@inria.fr

2013-8-6

http://sts.thss.tsinghua.edu.cn/Coqschool2013

Notes adapted from Assia Mahboubi (coq school 2010, Paris) and Benjamin Pierce (software foundations course, UPenn)
Recap

• easy proofs by simplification and reflexivity
• recursive types
• lists
• trees
• recursive definitions
Plan

• structural induction
• example 1: lists
• example 2: trees
Induction
Recursive types / structural induction (1/9)

Let us go back to the definition of list of days:

\[
\text{Inductive } \text{daylist} : \text{Type} := \\
\quad \text{nil} : \text{daylist} \mid \text{cons} : \text{day} \rightarrow \text{daylist} \rightarrow \text{daylist}.
\]

The \textbf{Inductive} keyword means that at definition time, this system generates an \textit{induction principle}:

\[
\text{daylist\_ind} : \forall \ P : \text{daylist} \rightarrow \text{Prop}, \\
\quad P \ \text{nil} \rightarrow \\
\quad (\forall \ (d : \text{day}) \ (l1 : \text{daylist}), \ P \ l1 \rightarrow P \ (\text{cons} \ d \ l1)) \rightarrow \\
\quad \forall \ l : \text{daylist}, \ P \ l
\]

Proof by Induction
For any \( P : daylist \rightarrow Prop \), to prove that the theorem
\[
\text{forall } l : \text{daylist}, \ P \ l
\]
holds, it is sufficient to:

- Prove that the property holds for the base case:
  - \( (P \ \text{nil}) \)
- Prove that the property is transmitted inductively:
  - \( \text{forall } (d : \text{day}) \ (l1 : \text{daylist}), \ P \ l1 \rightarrow P \ (d :: l1) \)

The type \text{daylist} is the smallest type containing \text{nil} and closed under cons.
Recursive types / structural induction (3/9)

The induction principles generated at definition time by the system allow to:

- Program by recursion (Fixpoint)
- Prove by induction (induction)

Example: append on lists.

Fixpoint app (l1 l2 : daylist) {struct l1} : daylist :=
match l1 with
| nil => l2
| d1 :: l1’ => d1 :: (app l1’ l2)
end.
Recursive types / structural induction (4/9)

**Associativity** of append on lists.

Theorem ass_app : forall l1 l2 l3 : daylist, l1 ++ (l2 ++ l3) = (l1 ++ l2) ++ l3.

Proof.
intros l1 l2 l3. induction l1 as [ | d1 l1' IHl1'].

[ ] ++ l2 ++ l3 = ([ ] ++ l2) ++ l3
- reflexivity.

\(d1 : \text{day}\)
\(l1' : \text{daylist}\)
\(l2 : \text{daylist}\)
\(l3 : \text{daylist}\)
\(IHl1' : l1' ++ l2 ++ l3 = (l1' ++ l2) ++ l3\)

= = = = = = = = = = = = = = = = = = = = = = = = =

\((d1 :: l1') ++ l2 ++ l3 = ((d1 :: l1') ++ l2) ++ l3\)
- simpl. rewrite IHl1'. reflexivity.

Qed.
Length of appended lists.

Fixpoint length (l:daylist) {struct l} : nat :=
  match l with
  | nil => 0
  | d :: t => S (length t)
end.

Theorem app_length : forall l1 l2 : daylist,
  length (l1 ++ l2) = (length l1) + (length l2).
Proof.
  intros l1 l2. induction l1 as [| d1 l1' IHl1'].
  - reflexivity.
  - simpl. rewrite IHl1'. reflexivity.
Qed.
Recursive types / structural induction (6/9)

Induction on natural numbers.

Lemma n_plus_zero : forall n:nat, n + 0 = n.
Proof.
  intros n. induction n as [ | n' IH].
  - reflexivity.
  - simpl. rewrite IH. reflexivity.
Qed.

Lemma n_plus_succ : forall n m :nat, n + S m = S (n + m).
Proof.
  intros n m. induction n as [ | n' IH].
  - reflexivity.
  - simpl. rewrite IH. reflexivity.
Qed.

Exercice 11  Show associativity and commutativity of +.
**Exercice 12** Show

\[ \text{length} \ (\text{alternate} \ l1 \ l2) = (\text{length} \ l1) + (\text{length} \ l2). \]

where

\[
\text{Fixpoint} \ \text{alternate} \ (l1 \ l2 : \text{daylist}) \ {\text{struct} \ l1 \ :} \ \text{daylist} := \\
\quad \text{match} \ l1 \ \text{with} \\
\quad \quad [\ ] \ \Rightarrow \ l2 \\
\quad \quad v1 :: l1' \ \Rightarrow \ \text{match} \ l2 \ \text{with} \\
\quad \quad \quad [\ ] \ \Rightarrow \ l1 \\
\quad \quad \quad v2 :: l2' \ \Rightarrow \ v1 :: v2 :: \text{alternate} \ l1' \ l2' \\
\quad \text{end} \\
\text{end.}
\]
Another recursive type: \textit{binary trees}.

Inductive \texttt{natBinTree} : Type :=
| \texttt{Leaf} : \texttt{nat} \to \texttt{natBinTree}
| \texttt{Node} : \texttt{nat} \to \texttt{natBinTree} \to \texttt{natBinTree} \to \texttt{natBinTree}.

\textbf{Abstract Syntax Trees for terms}.

Inductive \texttt{term} : Set :=
| \texttt{Zero} : \texttt{term}
| \texttt{One} : \texttt{term}
| \texttt{Plus} : \texttt{term} \to \texttt{term} \to \texttt{term}
| \texttt{Mult} : \texttt{term} \to \texttt{term} \to \texttt{term}. 
Recursive types / structural induction (9/9)

Counting leaves and nodes in binary trees.

Fixpoint count_leaves (t : natBinTree) {struct t} : nat :=
match t with
| leaf n => 1
| node n t1 t2 => (count_leaves t1) + (count_leaves t2)
end.

Fixpoint count_nodes (t : natBinTree) {struct t} : nat :=
match t with
| leaf n => 0
| node n t1 t2 => 1 + (count_nodes t1) + (count_nodes t2)
end.

Exercice 13 Show

Lemma leaves_and_nodes : forall t : natBinTree,
  count_leaves t = 1 + count_nodes t.