Inductive data types (II)

jean-jacques.levy@inria.fr
2013-8-6

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

5th Asian-Pacific Summer School on Formal Methods
August 5-10, 2013, Tsinghua University, Beijing, China

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:

```plaintext
Inductive daylist : Type :=
  nil : daylist | cons : day -> daylist -> daylist.
```

The **Inductive** keyword means that at definition time, this system generates an **induction principle**:

```plaintext
daylist_ind : forall P : daylist -> Prop,
  P nil ->
  (forall (d : day) (l1 : daylist), P l1 -> P (cons d l1)) ->
  forall l : daylist, P l
```

Recursive types / structural induction (2/9)

For any \( P : \text{daylist} \rightarrow \text{Prop} \), to prove that the theorem

```plaintext
forall l : daylist, P l
```

holds, it is sufficient to:

- Prove that the property holds for the base case:
  ```plaintext
  (P nil)
  ```
- Prove that the property is transmitted inductively:
  ```plaintext
  forall (d : day) (l1 : daylist),
  P l1 -> P (d :: l1)
  ```

The type `daylist` is the **smallest type** containing `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.

```plaintext
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**:

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

```plaintext
(d1 :: l1' ++ l2 ++ l3) = ((d1 :: l1') ++ l2) ++ l3  
```

- reflexivity.

```plaintext
d1 : day
l1' : daylist
l2 : daylist
l3 : daylist
IHl1' : l1' ++ l2 ++ l3 = (l1' ++ l2) ++ l3
```

```plaintext
(d1 :: l1') ++ l2 ++ l3 = ((d1 :: l1') ++ l2) ++ l3

- simpl. rewrite IHl1'. reflexivity.
Qed.
```
Recursive types / structural induction (5/9)

**Length** of appended lists.

Fixpoint length (l:daylist) : nat :=
\[\text{match } l \text{ with}
\begin{align*}
\text{| nil } & \Rightarrow 0 \\
\text{| d :: t } & \Rightarrow S (\text{length } t)
\end{align*}
\text{end.}
\]

Theorem app_length : forall l1 l2 : daylist,
\[\text{length (l1 ++ l2) } = \text{(length } l1 \text{) } + \text{(length } l2 \text{).}
\]
Proof.
\begin{align*}
\text{intros } l1 l2. \text{ induction } l1 \text{ as } [\text{| d1 l1' IHl1'}. \\
\text{- reflexivity.} \\
\text{- simpl. rewrite IHl1'. reflexivity.}
\end{align*}
Qed.

---

Recursive types / structural induction (6/9)

Induction on **natural** numbers.

Lemma n_plus_zero : forall n:nat, n + 0 = n.
Proof.
\begin{align*}
\text{intros } n. \text{ induction } n \text{ as } [\text{| n' IH}. \\
\text{- reflexivity.} \\
\text{- simpl. rewrite IH. reflexivity.}
\end{align*}
Qed.

Lemma n_plus_succ : forall n m :nat, n + S m = S (n + m).
Proof.
\begin{align*}
\text{intros } n m. \text{ induction } n \text{ as } [\text{| n' IH}. \\
\text{- reflexivity.} \\
\text{- simpl. rewrite IH. reflexivity.}
\end{align*}
Qed.

**Exercise 11** Show associativity and commutativity of +.

---

Recursive types / structural induction (7/9)

**Exercise 12** Show
\[\text{length } \text{alternate l1 l2} = \text{(length } l1 \text{) } + \text{(length } l2 \text{).}
\]
where
Fixpoint alternate (l1 l2 : daylist) : daylist :=
\[\text{match } l1 \text{ with}
\begin{align*}
\text{| [] } & \Rightarrow l2 \\
\text{| v1 : :l1' } & \Rightarrow \text{match } l2 \text{ with}
\begin{align*}
\text{| [] } & \Rightarrow \text{l1'} \\
\text{| v2 : :l2' } & \Rightarrow v1 :: v2 :: \text{alternate l1' l2'}
\end{align*}
\text{end.}
\end{align*}
\]

---

Recursive types / structural induction (8/9)

Another recursive type: **binary trees**.

Inductive natBinTree : Type :=
\[\text{| Leaf : nat } \rightarrow \text{natBinTree} \\
\text{| Node : nat } \rightarrow \text{natBinTree } \rightarrow \text{natBinTree } \rightarrow \text{natBinTree.}
\]

**Abstract Syntax Trees** for terms.

Inductive term : Set :=
\[\text{| Zero : term} \\
\text{| One : term} \\
\text{| Plus : term } \rightarrow \text{term } \rightarrow \text{term} \\
\text{| Mult : term } \rightarrow \text{term } \rightarrow \text{term.}
\]
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.

Exercise 13 Show
Lemma leaves_and_nodes : forall t : natBinTree,
  count_leaves t = 1 + count_nodes t.