5th Asian-Pacific Summer School on Formal Methods

August 5-10, 2013, Tsinghua University, Beijing, China

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)

Plan

- structural induction
- example 1: lists
- example 2: trees

Recap

- easy proofs by simplification and reflexivity
- recursive types
- lists
- trees
- recursive definitions

Induction

CENTRE DE RECHERCHE COMMUN
INRIA
MICROSOFT RESEARCH

Recursive types / structural induction (1/9)

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

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

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

```
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 $P : daylist \to Prop$, to prove that the theorem

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

holds, it is sufficient to:

- Prove that the property holds for the base case:
  - `(P nil)`
- Prove that the property is transmitted inductively:
  - ```
    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.

```
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 : day
l1' : daylist
l2 : daylist
l3 : daylist
IHl1' : l1' ++ l2 ++ l3 = (l1' ++ l2) ++ l3
============================
 (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) {struct l} : nat :=
  match l with
  | nil => O
  | 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 +.

Recursive types / structural induction (7/9)

Exercice 12 Show

```
    length (alternate l1 l2) = (length l1) + (length l2).
```

where

```
Fixpoint alternate (l1 l2 : daylist) {struct l1} : daylist :=
  match l1 with
  | [ ] => l2
  | v1 :: l1’ => match l2 with
     | [ ] => l1
     | v2 :: l2’ => v1 :: v2 :: alternate l1’ l2’
     end
  end.
```

Recursive types / structural induction (8/9)

Another recursive type: binary trees.

```
Inductive natBinTree : Type :=
| Leaf : nat -> natBinTree
| Node : nat -> natBinTree -> natBinTree -> natBinTree.
```

Abstract Syntax Trees for terms.

```
Inductive term : Set :=
| Zero : term
| One : term
| Plus : term -> term -> term
| Mult : term -> term -> 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.
```