General datatypes and algorithms

Yves Bertot

October 2018

Objectives

- Explore more varied data-types
  - More cases and more components
  - With dependency between the components
- Explore more general recursion

Trees

- We already saw how to represent a programming language
  1. As many cases as there kinds of instructions
  2. Each instruction may have several sub-components
- As a smaller example we show a data-type of binary trees

```
Inductive btree (T : Type) : Type :=
  Leaf | Node (val : T) (t1 t2 : btree T).
```

recursive programming on binary trees

```
Fixpoint rev_tree {T : Type} (t : btree T) : btree T :=
match t with
| Leaf => Leaf
| Node x t1 t2 => Node x (rev_tree t1) (rev_tree t2)
end.

Arguments Leaf {T}.
Arguments Node {T}.
```

recursive programming on binary trees (2)

```
Fixpoint count {T : Type} (p : T -> bool)
    (t : btree T) : nat :=
match t with
| Leaf => 0
| Note x t1 t2 =>
  (if p x then 1 else 0) + (count p t1 + count p t2)
end.

Definition size {T} (t : btree T) :=
  count (fun x => true) t.
```

- recursive calls are allowed on both sub-trees

Proof by induction on trees

- The same as for induction on natural numbers or lists
- Now there are two induction hypotheses
- Other kinds of “smaller” trees not easily handled

Example of a proof by induction

```
Lemma count_rev_tree {T} (p : T -> bool) t :
   count p (rev_tree t) = count p t.
Proof.
induction t as [ | a t1 IH1 t2 IH2].
2 subgoals
count p (rev_tree Leaf) = count p Leaf
easy.
count (rev_tree (Node a t1 t2)) = count p (Node a t1 t2)
simpl.
(if p a then 1 else 0) +
          (count p (rev_tree t2) + count p (rev_tree t1)) =
  (if p a then 1 else 0) + (count p t1 + count p t2)
```

Example of a proof by induction (2)

```
rewrite IH1.
  (if p a then 1 else 0) +
         (count p (rev_tree t2) + count p t1) =
  (if p a then 1 else 0) + (count p t1 + count p t2)
rewrite IH2.
  (if p a then 1 else 0) + (count p t2 + count p t1) =
  (if p a then 1 else 0) + (count p t1 + count p t2)
rewrite (Nat.add_comm (count p t2)).
  (if p a then 1 else 0) + (count p t1 + count p t2) =
  (if p a then 1 else 0) + (count p t1 + count p t2)
easy.
Qed.
```

Induction on the size of trees

```
Definition btree_size_ind
   forall P : btree T -> Prop,
   (forall t, (forall t', size t' < size t -> P t') -> P t) ->
   forall t, P t
  :=
  well_founded_induction
   (wf_inverse_image _ _ _ size PeanoNat.Nat.lt_wf_0).
```

- Now only one induction hypothesis
- But it can be used for any tree, even with different values
- The same pattern can be used for programming

Preparatory lemmas for bree_size_ind

```
Lemma size1 T (a : T) t1 t2 : size t1 < size (Node a t1 t2).
Proof.
unfold size; simpl.
unfold lt; apply Peano.le_n_S, Nat.le_add_r.
Qed.

Lemma size2 T (a : T) t1 t2 : size t2 < size (Node a t1 t2).
Proof.
unfold size; simpl.
unfold lt; apply Peano.le_n_S; rewrite Nat.add_comm; apply Nat.l
Qed.
```

A proof using btree_size_ind

```
Lemma redo_count_rev_tree T (p : T -> bool) t :
  count p (rev_tree t) = count p t.
Proof.
induction t as [t IH] using btree_size_ind.
destruct t as [ | a t1 t2].
  easy.
simpl.
rewrite IH.
  rewrite IH.
    rewrite (Nat.add_comm (count p t2)).
    easy.
  apply size1.
apply size2.
Qed.
```

Dependent components and certified data

- In pairs, the two components are independent
- Extension: sigma-types where the second components depends on the first one
- Example: bounded integers (n, h) where h is a proof that n is within bounds
- You need some for of "proof irrelevance"
- Used extensively in the *Mathematical Components* library
- Recursive programming on bounded integers is slightly more complex