Examples of inductive types in Coq

Daria Walukiewicz-Chrząszcz

02 april 2019

Trees

Inductive **nat_btree** : Set :=
| NLeaf : **nat_btree**
| NNode : **nat_btree** → **nat** → **nat_btree** → **nat_btree**.

Check nat_btree_ind.

nat_btree_ind
: ∀ P : **nat_btree** → Prop,
P NLeaf →
(∀ n : **nat_btree**, P n → ∀ ($n0$: **nat**) ($n1$: **nat_btree**),
P $n1$ → P (NNode n $n0$ $n1$)) →
∀ n : **nat_btree**, P n

Trees

```
Inductive nat_btree : Set :=
| NLeaf : nat_btree
| NNode : nat_btree → nat → nat_btree → nat_btree.
```

```
Check nat_btree_ind.
```

```
nat_btree_ind
     : ∀ P : nat_btree → Prop,
       P NLeaf →
       (∀ n : nat_btree, P n → ∀ (n0 : nat) (n1 : nat_btree),
        P n1 → P (NNode n n0 n1)) →
       ∀ n : nat_btree, P n
```

Mutually recursive types: odd_list and even_list

```
Inductive even_list : Set :=
| ENil : even_list
| ECons : nat → odd_list → even_list

with odd_list : Set :=
| OCons : nat → even_list → odd_list.
```

```
Check even_list_ind.

  even_list_ind
      : ∀ P : even_list → Prop,
        P ENil →
        (∀ (n : nat) (o : odd_list), P (ECons n o)) →
        ∀ e : even_list, P e
```

Mutually recursive types: odd_list and even_list

```
Inductive even_list : Set :=
| ENil : even_list
| ECons : nat → odd_list → even_list

with odd_list : Set :=
| OCons : nat → even_list → odd_list.
```

```
Check even_list_ind.
```

```
even_list_ind
    : ∀ P : even_list → Prop,
      P ENil →
      (∀ (n : nat) (o : odd_list), P (ECons n o)) →
      ∀ e : even_list, P e
```

Scheme — generation of induction principles

```
Scheme even_list_mut := Induction for even_list Sort Prop
with odd_list_mut := Induction for odd_list Sort Prop.
```

```
Check even_list_mut.
```

even_list_mut

$: \forall\ (P : \textbf{even\_list} \to \texttt{Prop})\ (P0 : \textbf{odd\_list} \to \texttt{Prop}),$

$P\ \text{ENil} \to$

$(\forall\ (n : \textbf{nat})\ (o : \textbf{odd\_list}),\ P0\ o \to P\ (\text{ECons}\ n\ o)) \to$

$(\forall\ (n : \textbf{nat})\ (e : \textbf{even\_list}),\ P\ e \to P0\ (\text{OCons}\ n\ e)) \to$

$\forall\ e : \textbf{even\_list},\ P\ e$

Scheme — generation of induction principles

```
Scheme even_list_mut := Induction for even_list Sort Prop
with odd_list_mut := Induction for odd_list Sort Prop.
```

```
Check even_list_mut.
```

even_list_mut

$: \forall\ (P : \textbf{even\_list} \to \texttt{Prop})\ (P0 : \textbf{odd\_list} \to \texttt{Prop}),$

$P\ \text{ENil} \to$

$(\forall\ (n : \textbf{nat})\ (o : \textbf{odd\_list}),\ P0\ o \to P\ (\text{ECons}\ n\ o)) \to$

$(\forall\ (n : \textbf{nat})\ (e : \textbf{even\_list}),\ P\ e \to P0\ (\text{OCons}\ n\ e)) \to$

$\forall\ e : \textbf{even\_list},\ P\ e$

Reflexive type: formula

```
Inductive formula : Set :=
| Eq : nat → nat → formula
| And : formula → formula → formula
| Forall : (nat → formula) → formula.
```

```
Check formula_ind.

  formula_ind
     : ∀ P : formula → Prop,
       (∀ n n0 : nat, P (Eq n n0)) →
       (∀ f0 : formula,
        P f0 → ∀ f1 : formula, P f1 → P (And f0 f1)) →
       (∀ f1 : nat → formula,
        (∀ n : nat, P (f1 n)) → P (Forall f1)) →
       ∀ f2 : formula, P f2
```

Reflexive type: `formula`

```
Inductive formula : Set :=
| Eq : nat → nat → formula
| And : formula → formula → formula
| Forall : (nat → formula) → formula.
```

```
Check formula_ind.

  formula_ind
      : ∀ P : formula → Prop,
        (∀ n n0 : nat, P (Eq n n0)) →
        (∀ f0 : formula,
         P f0 → ∀ f1 : formula, P f1 → P (And f0 f1)) →
        (∀ f1 : nat → formula,
         (∀ n : nat, P (f1 n)) → P (Forall f1)) →
        ∀ f2 : formula, P f2
```

Restrictions: positivity condition

```
Inductive term : Set :=
| App : term → term → term
| Abs : (term → term) → term.
```

```
Error: Non strictly positive occurrence of "term" in "(term
-> term) -> term"
```

Restrictions: positivity condition

Inductive **term** : Set :=
| App : **term** → **term** → **term**
| Abs : (**term** → **term**) → **term**.

```
Error: Non strictly positive occurrence of "term" in "(term
-> term) -> term"
```

Restrictions: only small inductive types in Set

Correct (definition of $\exists\, \phi \;\; P(\phi)$):

```
Inductive exProp (P:Prop->Prop) : Prop
:= exP_intro : forall X:Prop, P X -> exProp P.
```

Incorrect:

```
Inductive exSet (P:Set->Prop) : Set
:= exS_intro : forall X:Set, P X -> exSet P.
```

```
Error: Large non-propositional inductive types must be in Type.
```

Correct:

```
Inductive exType (P:Type->Prop) : Type
:= exT_intro : forall X:Type, P X -> exType P.
```

behind the scene:

$$\texttt{exType} : (P : \text{Type}_i \to \text{Prop}) \to \text{Type}_j$$

$$\texttt{exT\_intro} : \forall\, X : \text{Type}_k,\; P\, X -> \texttt{exType}\, P$$

where $k < j$ and $k < i$ (universe constraints)

Restrictions: only small inductive types in Set

Correct (definition of $\exists\, \phi \;\; P(\phi)$):

```
Inductive exProp (P:Prop->Prop) : Prop
:= exP_intro : forall X:Prop, P X -> exProp P.
```

Incorrect:

```
Inductive exSet (P:Set->Prop) : Set
:= exS_intro : forall X:Set, P X -> exSet P.
```

```
Error: Large non-propositional inductive types must be in Type.
```

Correct:

```
Inductive exType (P:Type->Prop) : Type
:= exT_intro : forall X:Type, P X -> exType P.
```

behind the scene:

$$\mathtt{exType} : (P : \mathsf{Type}_i \to \mathsf{Prop}) \to \mathsf{Type}_j$$

$$\mathtt{exT\_intro} : \forall\, X : \mathsf{Type}_k,\; P\, X -> \mathtt{exType}\, P$$

where $k < j$ and $k < i$ (universe constraints)

Restrictions: only small inductive types in Set

Correct (definition of $\exists\, \phi \;\; P(\phi)$):

```
Inductive exProp (P:Prop->Prop) : Prop
:= exP_intro : forall X:Prop, P X -> exProp P.
```

Incorrect:

```
Inductive exSet (P:Set->Prop) : Set
:= exS_intro : forall X:Set, P X -> exSet P.


Error: Large non-propositional inductive types must be in Type.
```

Correct:

```
Inductive exType (P:Type->Prop) : Type
:= exT_intro : forall X:Type, P X -> exType P.
```

behind the scene:

$$\texttt{exType} : (P : \texttt{Type}_i \to \texttt{Prop}) \to \texttt{Type}_j$$

$$\texttt{exT\_intro} : \forall\, X : \texttt{Type}_k,\; P\, X - > \texttt{exType}\, P$$

where $k < j$ and $k < i$ (universe constraints)

Definitional equality

```
Print "=".
```

Inductive **eq** $(A : \text{Type})\ (x : A) : A \to \text{Prop} := \text{eq\_refl} : x = x$

```
Check @eq_refl.
@eq_refl
    : forall (A : Type) (x : A), x = x
```

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
                                P x -> forall y : A, x = y -> P y
```

Definitional equality

```
Print "=".
```

Inductive **eq** $(A : \texttt{Type})\ (x : A) : A \to \texttt{Prop} :=$ eq_refl : $x = x$

```
Check @eq_refl.
@eq_refl
    : forall (A : Type) (x : A), x = x
```

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
                           P x -> forall y : A, x = y -> P y
```

Definitional equality

```
Print "=".
```

Inductive **eq** $(A : \texttt{Type})\ (x : A) : A \to \texttt{Prop} := \mathrm{eq\_refl} : x = x$

```
Check @eq_refl.
@eq_refl
    : forall (A : Type) (x : A), x = x
```

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
                         P x -> forall y : A, x = y -> P y
```

Definitional equality

```
Print "=".
```

Inductive **eq** $(A : \text{Type})$ $(x : A) : A \to$ Prop := eq_refl : $x = x$

```
Check @eq_refl.
@eq_refl
     : forall (A : Type) (x : A), x = x
```

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
                          P x -> forall y : A, x = y -> P y
```

reflexivity and rewrite

`reflexivity` $\equiv$ `apply eq_refl`

`rewrite H` $\equiv$ `apply eq_ind`

(where `H: a=b`)

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
                         P x -> forall y : A, x = y -> P y
```

reflexivity and rewrite

`reflexivity` $\equiv$ `apply eq_refl`

`rewrite H` $\equiv$ `apply eq_ind`

(where `H: a=b`)

```
eq_ind: forall (A : Type) (x : A) (P : A -> Prop),
                         P x -> forall y : A, x = y -> P y
```