

TD2: les entiers naturels

22 septembre 2009

En Coq, l'introduction d'un nouveau type de données s'effectue à l'aide d'un mécanisme de définition inductive qui ressemble beaucoup à la définition d'un type concret en Caml. Ainsi, le type `nat` des entiers naturels est introduit à l'aide de la définition inductive suivante :

```
Inductive nat : Set :=
  | 0 : nat
  | S : nat -> nat.
```

Cette définition ajoute à l'environnement courant trois nouvelles constantes :

- le type `nat : Set` (Set est le type des petits types) ;
- le constructeur `0 : nat` (constructeur constant) (lettre O en non zéro) ;
- le constructeur `S : nat -> nat` (constructeur à un argument de type `nat`).

Le système utilise ensuite le sucre syntaxique `0`, `1`, `2`, etc. pour désigner les entiers `0`, `S 0`, `S (S 0)`, `S (S (S 0))`, etc. La définition inductive ci-dessus engendre automatiquement un certain nombre de principes d'induction, dont le plus utilisé en pratique est le schéma de récurrence :

```
nat_ind :
  forall P : nat -> Prop,
    P 0 -> (forall n : nat, P n -> P (S n)) -> forall n : nat, P n
```

utilisé en interne par les tactiques `elim` et `induction`.

Exercice 1 : Addition

En Coq, l'addition est définie au moyen de la construction `Fixpoint`, qui est l'équivalent du "let rec" de Caml :

```
Fixpoint plus (n m:nat) {struct n} : nat :=
  match n with
  | 0 => m
  | S p => S (plus p m)
  end.
```

On notera la présence d'une annotation supplémentaire `{struct n}` qui indique explicitement l'argument de décroissance structurelle. Cette annotation permet à Coq de vérifier que les appels récursifs se font sur des arguments structurellement plus petits que `n`, et donc que la fonction ainsi définie termine toujours. Le système utilise la notation `n + m` pour désigner le terme `plus n m`.

1. Vérifier à l'aide des tactiques `simpl` et `reflexivity` qu'on a les égalités définitionnelles $0 + m = m$ et $S\ n + m = S\ (n + m)$. A-t-on les égalités définitionnelles $n + 0 = n$ et $n + S\ m = S\ (n + m)$?
2. Montrer les deux lemmes suivants :

$$\text{Lemma plus_n_0 : forall n, n + 0 = n.}$$

$$\text{Lemma plus_n_Sm : forall n m, n + S m = S (n + m).}$$
 On prouvera ces deux lemmes par récurrence sur n , à l'aide de la tactique `induction`.
3. Montrer que l'addition est commutative : `forall n m, n + m = m + n`.
4. Montrer que l'addition est associative : `forall n m p, (n + m) + p = n + (m + p)`.

Exercice 2 : Multiplication

En Coq, la multiplication est définie par

```
Fixpoint mult (n m:nat) {struct n} : nat :=
  match n with
  | 0 => 0
  | S p => m + mult p m
  end.
```

Le système utilise le sucre syntaxique $n * m$ pour désigner le terme `mult n m`.

1. Vérifier à l'aide des tactiques `simpl` et `reflexivity` qu'on a les égalités définitionnelles $0 * m = 0$ et $S\ n * m = m + n * m$.
2. Montrer que la multiplication est commutative et associative.
3. Montrer la distributivité : `forall n m p, (n + m) * p = n * p + m * p`.

Exercice 3 : Relation d'ordre

On peut définir la relation d'ordre usuelle sur les entiers `le : nat -> nat -> Prop` en posant :

$$\text{Definition le (n m : nat) := exists p, n + p = m.}$$

Cette définition n'est pas celle de la librairie standard de Coq, mais est bien entendu équivalente. Montrer que `le` est une relation d'ordre :

```
Lemma le_refl : forall n, le n n.
Lemma le_trans : forall n m p, le n m -> le m p -> le n p.
Lemma le_antisym : forall n m, le n m -> le m n -> n = m.
```

Vous pourrez utiliser `simpl`, `elim`, `induction`, `discriminate` et `generalize`.