

Preuves assistées par ordinateur – TD n° 5

Les entiers naturels en Coq

En Coq, l'introduction d'un nouv au typ d donné s' ff ctu à l'aid d'un mécanism d *définition inductive* qui r ss mbl b aucoup à la définition d'un typ concr t n Caml. Ainsi, l typ `n t` d s nti rs natur ls st introduit¹ à l'aid d la définition inductiv suivant :

```
Inductive n t : Set :=
| 0 : n t
| S : n t -> n t.
```

C tt définition ajout à l' nvironn m nt courant trois nouv ll s constant s :

- l typ `n t` : `Set` (`Set` st l typ d s p tits typ s);
- l construct ur `0` : `n t`² (construct ur constant);
- l construct ur `S` : `n t -> n t` (construct ur à un argum nt d typ `n t`).

L systèm utilis nsuit l suc syntaxiqu `0, 1, 2, 3, tc.` pour désign r l s nti rs `0, S 0, S (S 0), S (S (S 0))`, tc.

La définition inductiv ci-d ssus ng ndr automatiqu m nt un c rtain nombr d princip s d'induction, dont l plus utilisé n pratiqu st l schéma d récurr nc

```
n t_ind :
  for ll P : n t -> Prop,
    P 0 -> (for ll n : n t, P n -> P (S n)) -> for ll n : n t, P n
```

utilisé n int rn par l s tactiqu s `elim t induction`.

Exercice 1 – L'addition En Coq, l'addition³ st défini au moy n d la construction `Fixpoint`, qui st l'équival nt du « `let rec` » d Caml :

```
Fixpoint plus (n m:n t) : n t :=
  m tch n with
  | 0 => m
  | S p => S (plus p m)
end.
```

Il st important d not r qu l s app ls récursifs s font ici sur un pr mi r argum nt `n` d plus `n` plus p tit. Il s'agit n fait d *décroissance structurelle*. Coq r fus l s définitions pour l squ ls il n' st pas n m sur d vérifi r c tt propriété, t qui risqu nt donc d n pas forcém nt t rmin r.⁴

L systèm utilis la notation `n + m` pour désign r l t rm plus `n m`.

1. cf fichier `theories/Init/Datatypes.v` ou page <http://coq.inria.fr/library-eng.html>

2. Attention! le constructeur s'appelle `0` (lettre « O ») et non `0` (« zéro »).

3. cf fichier `theories/Init/Peano.v`

4. Dans les versions de Coq antérieures à 8.2, il fallait indiquer l'argument de décroissance à l'aide d'une annotation `{struct n}`.

1. Vérifier à l'aide des tactiques `simpl` et `reflexivity` qu'on a les égalités définies

$$0 + m = m \quad \text{et} \quad S\ n + m = S\ (n + m).$$

A-t-on les égalités définies $n + 0 = n$ et $n + S\ m = S\ (n + m)$?

2. Montrer les deux lemmes suivants :

```
Lemm plus_n_0 : for ll n,      n + 0 = n.
Lemm plus_n_Sm : for ll 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 associative : `for ll n m p, (n + m) + p = n + (m + p)`.
4. Montrer que l'addition est commutative : `for ll n m, n + m = m + n`.

Exercice 2 – La multiplication En Coq, la multiplication est définie par

```
Fixpoint mult (n m:nat) : nat :=
  m tch 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 définies

$$0 * m = 0 \quad \text{et} \quad S\ n * m = m + n * m.$$

2. Prouver les égalités concernant $m * 0$ et $n * S\ m$.
3. Montrer la propriété de distributivité : `for ll n m p, (n + m) * p = n * p + m * p`.
4. Montrer que la multiplication est commutative et associative.

Exercice 3 – La relation d'ordre On peut définir⁵ la relation d'ordre usuelle sur les entiers

`le : nat -> nat -> Prop` en posant :

```
Definition le (n m : nat) := exists p, n + p = m.
```

Montrer que `le` est une relation d'ordre :

```
Lemm le_refl :      for ll n,      le n n.
Lemm le_trans :    for ll n m p,  le n m -> le m p -> le n p.
Lemm le_antisym :  for ll n m,    le n m -> le m n -> n = m.
```

Tactiques utiles : `simpl`, `elim`, `induction`, `rewrite`, `discriminate`, `injection`, `f_equal`.

5. Cette définition n'est pas celle de la librairie standard de Coq (cf fichier `theories/Init/Peano.v`), mais est bien entendu équivalente (prouvez-le!).