

INE22 : Preuves de Programmes – TP n° 1

Premiers pas en Coq

Pour résoudre les exercices, on s'aidera de la correspondance suivante

Tactique	Règle(s) de déduction
assumption	Axiome
intro, intros	\Rightarrow -intro, \forall -intro, \neg -intro
apply	\Rightarrow -élim, \forall -élim, \neg -élim
split	\wedge -intro, \top -intro
left, right	\forall -intro ₁ , \forall -intro ₂
exists	\exists -intro
destruct	\wedge -gauche, \forall -gauche, \perp -gauche, \top -gauche, \exists -gauche

et de la documentation en ligne : <http://www.pps.jussieu.fr/~miquel/enseignement/mpri/>

Exercice 1 (Calcul propositionnel) Établir en Coq les formules suivantes :

1. `forall A : Prop, A -> A`
2. `forall A B C : Prop, (A -> B) -> (B -> C) -> A -> C`
3. `forall A B : Prop, A /\ B <-> B /\ A`
4. `forall A B : Prop, A \/ B <-> B \/ A`

Exercice 2 (Calcul des prédicats) Après avoir effectué les déclarations suivantes

```
Parameter X Y : Set.
Parameter A B : X -> Prop.
Parameter R : X -> Y -> Prop.
```

établir en Coq les formules suivantes :

1. `(forall x : X, A x /\ B x) <-> (forall x : X, A x) /\ (forall x : X, B x)`
2. `(exists x : X, A x \/ B x) <-> (exists x : X, A x) \/ (exists x : X, B x)`
3. `(forall x : X, exists y : Y, R x y) -> (exists y : Y, forall x : X, R x y)`
4. `(exists y : Y, forall x : X, R x y) -> (forall x : X, exists y : Y, R x y)`

Exercice 3 (Relations d'ordre) En Coq, on considère un type $E : \text{Set}$ muni d'une relation binaire R dont on suppose qu'elle satisfait aux axiomes des relations d'ordre :

```
Parameter E : Set.
Parameter R : E -> E -> Prop.
```

```
Axiom refl : forall (x : E), R x x
Axiom trans : forall (x y z : E), R x y -> R y z -> R x z.
Axiom antisym : forall (x y : E), R x y -> R y x -> x = y.
```

On définit les notions de plus petit élément et d'élément minimal de la façon suivante :

```
Definition smallest (x0 : E) := forall (x : E), R x0 x.
```

```
Definition minimal (x0 : E) := forall (x : E), R x x0 -> x = x0.
```

Quels sont les types des objets `smallest` et `minimal` ?

Énoncer en Coq puis démontrer les lemmes suivants :

1. Si R admet un plus petit élément, alors celui-ci est unique.
2. Le plus petit élément, s'il existe, est un élément minimal.
3. Si R admet un plus petit élément, alors il n'y a pas d'autre élément minimal que celui-ci.

Indications : En Coq, une définition s'utilise en remplaçant le *definiendum* par son *definiens* à l'aide de la tactique `unfold <definiendum>` (*unfold* = déplier). L'égalité se traite à l'aide des tactiques `reflexivity`, `symmetry`, `transitivity <terme>` et `rewrite <hypothèse>`.

Exercice 4 – Les entiers 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 introduit¹ 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 : nat2` (constructeur constant) ;
- 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 5 – L'addition En Coq, l'addition³ 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.
```

¹cf fichier `theories/Init/Datatypes.v`

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

³cf fichier `theories/Init/Peano.v`

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

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

A-t-on les égalités définitionnelles `n + 0 = n` et `n + S m = S (n + m)` ?

2. Montrer les deux lemmes suivants :

`Lemma plus_n_0 : forall n, n + 0 = n.`

`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 6 – La 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

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

2. Montrer que la multiplication est commutative et associative⁴.
3. Montrer la propriété de distributivité à droite : `forall n m p, (n + m) * p = n * p + m * p.`

Exercice 7 – 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 :

`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.`

Tactiques utiles : `simpl, elim, induction, generalize.`

⁴Indication : Montrer d'abord la distributivité à gauche

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