Initiation aux preuves formelles : Partiel écrit

17 décembre 2024 — Sergueï Lenglet et Pierre Rousselin

Consignes et conseils :

- Durée 2h00. Un résumé de cours manuscrit est autorisé.
- Le barème est donné à titre indicatif et est susceptible d'être modifié. Le total des points avant modification éventuelle du barème est 32. La note sera ramenée sur 20.
- Les exercices sont indépendants les uns des autres.
- La clarté des réponses et la propreté de la copie seront des éléments importants de l'évaluation.
- En annexe sont rappelés certains lemmes qui peuvent être utilisés dans certains exercices.

Exercice 1 : Effet des tactiques sur un état de preuve (8 points)

1. On rappelle les 4 résultats suivants sur les nombres réels :

```
add_eq_compat_l : forall r r1 r2 : R, r1 = r2 -> r + r1 = r + r2
add_eq_compat_r : forall r r1 r2 : R, r1 = r2 -> r1 + r = r2 + r
add_eq_reg_l : forall r r1 r2 : R, r + r1 = r + r2 -> r1 = r2
add_eq_reg_r : forall r r1 r2 : R, r1 + r = r2 + r -> r1 = r2
```

Dans chacun des états de preuve suivants, dire comment est modifié l'état de la preuve par la tactique donnée, ou bien si la tactique provoque une erreur, auquel cas en donner la raison.

a)
```
x, y : R
H : x + x = y + y
======================== (1 / 1)
x = y
```
```
apply (add_eq_reg_l x).
```

b)
```
x, y : R
H : x + x = y + y
======================== (1 / 1)
x = y
```
```
apply add_eq_reg_l.
```

c)
```
x, y : R
H : x = y
======================== (1 / 1)
x + y = x + x
```
```
rewrite (add_eq_reg_l x y x).
```

d)
```
x : R
H : x + x = x + 0
======================== (1 / 1)
x = 0
```
```
apply add_eq_reg_l in H.
```

2. On travaille maintenant sur les entiers de type `nat`, et les tactiques `simpl` et `discriminate`. On rappelle la définition de la multiplication des entiers naturels :

```
Fixpoint mul n m =
  match n with
  | 0 => 0
  | S p => m + mul p m
  end.
```

Dans chacun des cas suivants dire l'effet de la tactique donnée sur l'état de preuve donné. En cas d'erreur, expliquer cette erreur.

a)
```
n : nat
H : n = 21
======================== (1 / 1)
2 * n = 42
```

```
   simpl.
```

b)
```
n : nat
H : n = 21
======================== (1 / 1)
n * 2 = 42
simpl.
```

c)
```
n : nat
H : 1 = S (S n)
======================== (1 / 1)
2 + 2 = 5
discriminate H.
```

d)
```
n : nat
H : 0 * n = 0
======================== (1 / 1)
2 = 2
discriminate H.
```

--- * ---

Exercice 2 : Prouvable ou non ? (8 points)

Pour chacun des états de preuve suivants, dire s'ils sont prouvables ou non. Dans le cas où il sont prouvables donner une preuve en Coq, dans le cas où ils ne sont pas prouvable, donner un contre-exemple, c'est-à-dire des propositions ou des valeurs de variables rendant vrai le contexte mais pas le but. Pour prouver les lemmes prouvables, vous pouvez utiliser n'importe quel lemme donné à la fin du sujet.

1.
```
A, B, C, D : Prop
H1 : A \/ B
H2 : A -> C
H3 : B -> D
H4 : C -> D
======================== (1 / 1)
D
```

2.
```
======================== (1 / 1)
exists n : nat, n + n = n
```

3.
```
========================== (1 / 1)
forall n : nat, exists p : nat, n = S p
```

4. Pour cette question, on pourra utiliser l'axiome du tiers exclu, que voici :

```
classic : forall P : Prop, P \/ ~ P
```

Rappel : en Coq, si `P` est une proposition, `~P` est une notation pour `P -> False`.

```
P : Prop
H : ~ ~ P
========================== (1 / 1)
P
```

--- * ---

Exercice 3 : Ordre sur les réels et preuve mathématique détaillée (8 points)

1. On démontre ici le théorème suivant : soit f une fonction affine strictement croissante, alors son coefficient directeur est strictement positif. On suppose connus seulement les résultats suivants :

```
lt_0_1 : 0 < 1
mul_0_l : forall r : R, 0 * r = 0
mul_0_r : forall r : R, r * 0 = 0
mul_1_l : forall r : R, 1 * r = r
mul_1_r : forall r : R, r * 1 = r
mul_lt_compat_l : forall r r1 r2 : R, 0 < r -> r1 < r2 -> r * r1 < r * r2
mul_lt_compat_r : forall r r1 r2 : R, 0 < r -> r1 < r2 -> r1 * r < r2 * r
mul_lt_reg_l : forall r r1 r2 : R, 0 < r -> r * r1 < r * r2 -> r1 < r2
mul_lt_reg_r : forall r r1 r2 : R, 0 < r -> r1 * r < r2 * r -> r1 < r2
```

```
add_0_l : forall r : R, 0 + r = r
add_0_r : forall r : R, r + 0 = r
add_comm : forall r1 r2, r1 + r2 = r2 + r1
add_lt_compat_l : forall r r1 r2, r1 < r2 -> r + r1 < r + r2
add_lt_compat_r : forall r r1 r2, r1 < r2 -> r1 + r < r2 + r
add_lt_reg_l : forall r r1 r2 : R, r + r1 < r + r2 -> r1 < r2
add_lt_reg_r : forall r r1 r2 : R, r1 + r < r2 + r -> r1 < r2
```

Recopier sur la copie et annoter la preuve suivante pour y faire apparaître les applications, réécritures (ou définitions) permettant de justifier chaque étape.

Démonstration. Soient a et b tels que pour tout réel x, $f(x) = ax + b$ (hypothèse (E)). Comme f est supposée croissante, on a, pour tous réels x et y, $x < y \implies f(x) < f(y)$ (hypothèse (H)). On sait que $0 < 1$. Or,

$$\begin{array}{c} 0 < 1 \\ \Downarrow \\ f(0) < f(1) \\ \Downarrow \\ a \times 0 + b < a \times 1 + b \\ \Downarrow \\ a \times 0 < a \times 1 \\ \Downarrow \\ 0 < a \times 1 \\ \Downarrow \\ 0 < a \end{array}$$

□

2. On suppose connues par Coq les définitions suivantes :

```
Definition increasing (f : R -> R) := forall (x y : R), x < y -> f x < f y.
Definition aff (a b : R) (x : R) := a * x + b.
```

En résumé, `(aff a b)` est la fonction affine qui à x associe $ax+b$. Écrire la preuve en Coq du lemme suivant :

```
Lemma aff_incr_a_pos (a b : R) : increasing (aff a b) -> 0 < a.
```

3. Écrire la preuve mathématique très détaillée du résultat suivant : toute fonction affine de coefficient directeur strictement positif est strictement croissante.

4. Écrire la preuve en Coq du lemme :

```
Lemma a_pos_aff_incr (a b : R) : 0 < a -> increasing (aff a b).
```

--- * ---

Exercice 4 : Opération `ordre` sur `nat`, à valeurs dans un type à trois éléments (8 points)

Dans cet exercice, on se donne le type `Comparaison` qui est un type inductif à 3 éléments :

```
Inductive Comparaison := Lt | Eq | Gt.
```

Intuitivement, `Lt` signifie « *less than* », (strictement) inférieur, `Eq` signifie « *equal* », égal, et `Gt` signifie « *greater than* », (strictement) supérieur. On définit, d'une certaine manière, la fonction `ordre : nat -> Comparaison` dont on connaît les 4 propriétés suivantes :

```
ordre_0_0: ordre 0 0 = Eq
ordre_0_succ: forall m : nat, ordre 0 (S m) = Lt
ordre_succ_0: forall n : nat, ordre (S n) 0 = Gt
ordre_succ_succ: forall n m : nat, ordre (S n) (S m) = ordre n m
```

1. En utilisant ces 4 règles, évaluer `ordre 5 2`, `ordre 1 3` et `ordre 2 2`.

2. Prouver en Coq le lemme suivant :
```
Lemma ordre_diag (n : nat) : ordre n n = Eq.
```

3. Prouver en Coq le lemme suivant :
```
Lemma ordre_succ_diag_r (n : nat) : ordre n (S n) = Lt.
```

4. On **admet** le résultat suivant :
```
ordre_Lt_trans: forall (n m p : nat), ordre n m = Lt -> ordre m p = Lt -> ordre n p = Lt.
```
Prouver en Coq le lemme suivant :
```
Lemma ordre_add_r (n p : nat) : ordre n ((S p) + n) = Lt.
```

--- * ---

Liste de lemmes qu'on peut utiliser

```
add_0_l : forall n : nat, 0 + n = n
add_0_r : forall n : nat, n + 0 = n
add_1_l : forall n : nat, 1 + n = S n
add_1_r : forall n : nat, n + 1 = S n
add_assoc : forall n m p : nat, (n + m) + p = n + (m + p)
add_comm : forall n m : nat, n + m = m + n
add_succ_l : forall n m : nat, S n + m = S (n + m)
add_succ_r : forall n m : nat, n + S m = S (n + m)
lt_0_1 : 0 < 1
mul_0_l : forall r : R, 0 * r = 0
mul_0_r : forall r : R, r * 0 = 0
mul_1_l : forall r : R, 1 * r = r
mul_1_r : forall r : R, r * 1 = r
mul_lt_compat_l : forall r r1 r2 : R, 0 < r -> r1 < r2 -> r * r1 < r * r2
mul_lt_compat_r : forall r r1 r2 : R, 0 < r -> r1 < r2 -> r1 * r < r2 * r
mul_lt_reg_l : forall r r1 r2 : R, 0 < r -> r * r1 < r * r2 -> r1 < r2
mul_lt_reg_r : forall r r1 r2 : R, 0 < r -> r1 * r < r2 * r -> r1 < r2
add_0_l : forall r : R, 0 + r = r
add_0_r : forall r : R, r + 0 = r
add_comm : forall r1 r2, r1 + r2 = r2 + r1
add_lt_compat_l : forall r r1 r2, r1 < r2 -> r + r1 < r + r2
add_lt_compat_r : forall r r1 r2, r1 < r2 -> r1 + r < r2 + r
add_lt_reg_l : forall r r1 r2 : R, r + r1 < r + r2 -> r1 < r2
add_lt_reg_r : forall r r1 r2 : R, r1 + r < r2 + r -> r1 < r2
```