INITIATION AUX PREUVES FORMELLES : SUJET 0
novembre 2024 — Sergueï Lenglet et Pierre Rousselin

Ce document, non contractuel a pour but de vous faire travailler une épreuve dont le format et, dans une moindre mesure, le contenu devrait être proche de celui du partiel sur papier.

Exercice 1 : Effet des tactiques sur un état de preuve

1. **rewrite** ou **apply**? On rappelle les 4 résultats suivants sur les nombres réels :

`Rplus_eq_compat_l` : `forall r r1 r2 : R, r1 = r2 -> r + r1 = r + r2`

`Rplus_eq_compat_r` : `forall r r1 r2 : R, r1 = r2 -> r1 + r = r2 + r`

`Rplus_eq_reg_l` : `forall r r1 r2 : R, r + r1 = r + r2 -> r1 = r2`

`Rplus_eq_reg_r` : `forall r r1 r2 : R, r1 + r = r2 + r -> r1 = r2`

Dans chacun des contextes suivants, dire l'effet de la tactique donnée ; si la tactique peut fonctionner en donnant plus d'arguments, les donner et dire comment l'état de la preuve est transformé.

a) `x : R`
`H : x + x = x`
 ===== (1 / 1)
`x = 0`

`apply (Rplus_eq_reg_l x).`

b) `x : R`
`H : x + x = x`
 ===== (1 / 1)
`x = 0`

`rewrite (Rplus_eq_reg_l x).`

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

`apply Rplus_eq_compat_l.`

d) `x : R`
`H : x + x = x + 0`
 ===== (1 / 1)
`x = 0`

`apply Rplus_eq_reg_l in H.`

2. On travaille maintenant sur les `nat`, et les tactiques `simpl` et `discriminate`. On rappelle la définition de l'addition sur les entiers naturels :

```
Fixpoint add n m =
  match n with
  | 0 => m
  | S p => S (add 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 = 42`
 ===== (1 / 1)
`2 + n = 44`

`simpl.`

b) `n : nat`
`H : n = 42`
 ===== (1 / 1)
`n + 2 = 44`

`simpl.`

```

c) n : nat
   H : S (S n) = 1
   ===== (1 / 1)
   42 = 43
   discriminate H.

d) n : nat
   H : S n = 1
   ===== (1 / 1)
   42 = 43
   discriminate H.

e) n : nat
   H : 3 + n = 0
   ===== (1 / 1)
   42 = 43
   discriminate H.

```

--- * ---

Exercice 2 : Prouvable ou non ?

Pour chacun des états de preuve suivant, dire s'ils sont prouvables ou non. Dans le cas où il sont prouvables donner une preuve en Coq, dans le cas où il ne sont pas prouvables, 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 (ou d'autres, en demandant aux enseignants si c'est d'accord).

<pre> 1. A, B, C : Prop H1 : A \ / B H2 : A -> C ===== (1 / 1) C 2. n : nat H : exists k : nat, n = k + k + k + k ===== (1 / 1) exists q : nat, n = 2 * q </pre>	<pre> 3. n : nat A, B : Prop H1 : B H2 : A -> ~ B H3 : A ===== (1 / 1) n = 42 4. ===== (1 / 1) ~ (forall n : nat, exists k : nat, n = 2 * k) </pre>
---	--

--- * ---

Exercice 3 : La puissance !

On définit l'opération puissance sur les entiers naturels de la façon suivante.

```

Fixpoint pow n m :=
  match m with
  0 => 1 (* n ^ 0 = 1 *)
  | (S m') => n * pow n m' (* n ^ (m + 1) = n * (n ^ m) *)
  end.

```

Notation "n ^ m" := (pow n m).

- Donner les différentes étapes de calcul pour évaluer 5^2 et 10^0 (on évaluera directement les multiplications à la fin du calcul).
- Dans cette partie, vous pourrez utiliser les résultats suivants :


```

add_comm : forall n m : nat, n + m = m + n
add_assoc : forall n m p : nat, n + (m + p) = n + m + p
add_0_r : forall n : nat, n + 0 = n
add_succ_r : forall n m : nat, n + S m = S (n + m)
mul_1_r : forall n : nat, n * 1 = n
mul_assoc : forall n m p : nat, n * (m * p) = n * m * p
mul_comm : forall n m : nat, n * m = m * n

```

Donner des preuves en Coq des résultats suivants :

- a) `Lemma pow_2_square (n : nat) : n ^ 2 = n * n.`
- b) `Lemma pow_1_1 (m : nat) : 1 ^ m = 1.`
- c) `Lemma pow_add_r (m n k : nat) : m ^ (n + k) = (m ^ n) * (m ^ k).`

--- * ---

Exercice 4 : Ordre sur les réels et preuve mathématique détaillée

1. On démontre ici le théorème suivant : pour tous réels r_1, r_2, r_3 et r_4 , si $r_1 < r_2$ et $r_3 < r_4$, alors

$$r_1 + r_3 < r_2 + r_4.$$

On suppose connus seulement les résultats suivants :

`Rlt_trans : forall r1 r2 r3 : R, r1 < r2 -> r2 < r3 -> r1 < r3`

`Rplus_lt_compat_l : forall r r1 r2, r1 < r2 -> r + r1 < r + r2`

`Rplus_lt_compat_r : forall r r1 r2, r1 < r2 -> r1 + r < r2 + r`

`Rplus_comm : forall r1 r2, r1 + r2 = r2 + r1`

Annoter la preuve suivante pour y faire apparaître les applications et réécritures permettant de justifier chaque étape.

Démonstration. Soient r_1, r_2, r_3 et r_4 quatre réels. On suppose que $r_1 < r_2$ et $r_3 < r_4$. Or,

$$\begin{array}{ccc} r_1 < r_2 & & r_3 < r_4 \\ \Downarrow & & \Downarrow \\ r_1 + r_3 < r_2 + r_3 & = & r_2 + r_3 < r_2 + r_4 \\ & \Downarrow & \\ & r_1 + r_3 < r_2 + r_4. & \square \end{array}$$

2. Écrire la preuve en Coq du lemme suivant :

`Lemma Rplus_lt_compat :`

`forall r1 r2 r3 r4 : R, (r1 < r2) -> (r3 < r4) -> (r1 + r3 < r2 + r4).`

3. Écrire la preuve mathématique très détaillée du résultat suivant : la fonction qui à un réel associe son opposé est strictement décroissante sur $\mathbb{R}$. Vous pouvez supposés connus, en plus des lemmes précédents, les résultats suivants :

`Rplus_opp_l : forall r : R, - r + r = 0`

`Rplus_assoc : forall r1 r2 r3 : R, r1 + r2 + r3 = r1 + (r2 + r3)`

`Rplus_opp_r : forall r : R, r + - r = 0`

`Rplus_0_l : forall r : R, 0 + r = r`

`Rplus_0_r : forall r : R, r + 0 = r`

4. Écrire la preuve en Coq du lemme :

`Lemma Ropp_lt_contravar : forall r1 r2 : R, r2 < r1 -> - r1 < - r2.`

--- * ---

Exercice 5 : Deux sens de « suite croissante »

Definition `croissante (Un : nat -> R) := forall n, Un n <= Un (S n).`

On veut prouver le lemme suivant.

`Lemma croissante_croissante (Un : nat -> R) : croissante Un ->`

`forall n m, (n <= m)%nat -> (Un n <= Un m).`

1. Écrire soigneusement la démonstration mathématique par récurrence *sur m*.
2. Écrire la preuve en Coq. La correction utilise les lemmes `Nat.le_0_r`, `Nat.le_succ_r`, `Rle_refl` et `Rle_trans`.

--- * ---

Liste de lemmes utilisés pour préparer ce sujet

```
add_0_r : forall n : nat, (n + 0)%nat = n
add_assoc : forall n m p : nat, (n + (m + p))%nat = (n + m + p)%nat
add_comm : forall n m : nat, (n + m)%nat = (m + n)%nat
le_0_r : forall n : nat, (n <= 0)%nat <-> n = 0%nat
le_succ_r : forall n m : nat, (n <= S m)%nat <-> (n <= m)%nat \ / n = S m
mul_1_l : forall n : nat, (1 * n)%nat = n
mul_1_r : forall n : nat, (n * 1)%nat = n
mul_add_distr_l : forall n m p : nat, (n * (m + p))%nat = (n * m + n * p)%nat
mul_assoc : forall n m p : nat, (n * (m * p))%nat = (n * m * p)%nat
mul_succ_l : forall n m : nat, (S n * m)%nat = (n * m + m)%nat
mul_succ_r : forall n m : nat, (n * S m)%nat = (n * m + n)%nat
Rle_refl : forall r : R, r <= r
Rle_trans : forall r1 r2 r3 : R, r1 <= r2 -> r2 <= r3 -> r1 <= r3
Rplus_0_l : forall r : R, 0 + r = r
Rplus_0_r : forall r : R, r + 0 = r
Rplus_assoc : forall r1 r2 r3 : R, r1 + r2 + r3 = r1 + (r2 + r3)
Rplus_eq_compat_l : forall r r1 r2 : R, r1 = r2 -> r + r1 = r + r2
Rplus_eq_reg_l : forall r r1 r2 : R, r + r1 = r + r2 -> r1 = r2
Rplus_lt_compat_l : forall r r1 r2 : R, r1 < r2 -> r + r1 < r + r2
Rplus_lt_compat_r : forall r r1 r2 : R, r1 < r2 -> r1 + r < r2 + r
Rplus_opp_l : forall r : R, - r + r = 0
```