INITIATION AUX PREUVES FORMELLES : TD 1
septembre 2024 — Pierre Rousselin

1 Logique de base

Exercice 1 : 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 A, B et C qui rendent le contexte vrai mais pas le but.

<p>1. A, B, C : Prop
H1 : A $\vee$ B
H2 : A $\rightarrow$ C
H3 : B $\rightarrow$ C
===== (1 / 1)
B</p>	<p>3. A, B, C : Prop
H1 : A $\vee$ B
H2 : A $\rightarrow$ C
H3 : B $\rightarrow$ C
===== (1 / 1)
A $\wedge$ C</p>
<p>2. A, B, C : Prop
H1 : A $\vee$ B
H2 : A $\rightarrow$ C
H3 : B $\rightarrow$ C
===== (1 / 1)
C</p>	<p>4. A, B, C : Prop
H1 : A $\vee$ B
H2 : A $\rightarrow$ C
H3 : B $\rightarrow$ C
===== (1 / 1)
(A $\wedge$ C) $\vee$ (B $\wedge$ C)</p>
--- * ---	

2 Entiers naturels et calcul

On rappelle la définition usuelle de `nat` en Coq :

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

On dit que `0` (la lettre O) et `S` sont les deux *constructeurs* du type `nat`. En pratique, on utilise la notation `0` (le chiffre 0) au lieu de `0` et `1, 2, 3, ...` pour `S 0, S (S 0), S (S (S 0)), ...`. Cela signifie beaucoup de choses¹ :

- `0` est un entier naturel.
- Si `n` est un entier naturel, alors son successeur `S n` est encore un entier naturel.
- Tout entier est égal à `0` ou au successeur d'un entier.
- `0` n'est jamais égal au successeur d'un entier naturel (les images de deux constructeurs distinctes sont toujours distinctes).
- Deux entiers qui ont le même successeur sont égaux.

Exercice 2 : La tactique `discriminate`

Si l'hypothèse H est une égalité entre deux naturels, la tactique `discriminate` H essaie de prouver `False` (et donc de terminer la preuve) en simplifiant H jusqu'à trouver une égalité du type `0 = S n` ou `S n = 0`. Si ce n'est pas possible, le message d'erreur *not a discriminable equality* est émis.

Que se passe-t-il dans chacun des cas suivant lorsque `discriminate` H est utilisé ? Là où elles apparaissent, les variables `n` et `m` désignent des entiers naturels quelconques.

1. H : S n = 0

1. Pour les curieux, ce sont certains des axiomes de Peano, https://fr.wikipedia.org/wiki/Axiomes_de_Peano. On appelle donc cette formalisation des entiers naturels « entiers de Peano ».

2. $H : S (S n) = S 0$
3. $H : S n = S (S m)$
4. $H : 2 = S (S (S n))$

--- * ---

Exercice 3 : Calculs

Certaines opérations sont définies directement à l'aide d'un programme, c'est le cas de l'addition sur `nat` :

```
Fixpoint add (n m : nat) :=
match n with
| 0 => m (* 0 + m = m, règle add_0_l *)
| S p => S (add n m) (* (S p) + m = S (p + m), règle add_succ_l *)
end.
```

Muni de ce programme, Coq peut évaluer (réduire) certaines expressions comportant des additions. En détaillant les calculs (utiliser le nom des « règles » données en commentaire), donner le résultat des calculs suivants. Là où elles apparaissent, les variables `n` et `m` désignent des entiers naturels quelconques.

1. `Eval simpl in 2 + 2.`
2. `Eval simpl in 0 + 3.`
3. `Eval simpl in 3 + 0.`
4. `Eval simpl in (S (S n)) + m.`
5. `Eval simpl in n + (S m).`
6. `Eval simpl in (0 + (S n)) + 0.`

--- * ---

Exercice 4 : Induction

Compléter la preuve suivante, d'abord en Coq, puis en langage naturel très détaillé.

```
Lemma add_0_r (n : nat) : n + 0 = n.
Proof.
induction n as [| n IH].
- (* compléter *)
- (* compléter *)
Qed.
```

--- * ---

Exercice 5 : double

On considère le développement Coq suivant :

```
Fixpoint double (n : nat) :=
match n with
| 0 => 0
| S n => S (S (double n))
end.
Lemma double_0 : double 0 = 0.
Proof. (* à compléter 1 *) Qed.
Lemma double_succ (n : nat) : double (S n) = S (S (double n)).
Proof. (* à compléter 1 *) Qed.
Lemma add_diag (n : nat) : n + n = double n.
Proof. (* à compléter 2 *) Qed.
```

1. Quelle est la preuve commune de `double_0` et `double_succ` ?
2. Prouver `add_diag`. Vous avez besoin d'une propriété supplémentaire sur l'addition, laquelle ?

--- * ---