LEÇON 3 : Logique du Premier Ordre et Égalité

Quantificateurs universels, existentiels et règles équationnelles

Pablo Donato

31/03/2026

Après avoir exploré la logique propositionnelle, nous passons aujourd'hui à la logique des prédicats. Nous allons voir comment Rocq gère les quantificateurs universels ($\forall$) et existentiels ($\exists$), ainsi que la notion fondamentale d'égalité.

Dans Rocq, les prédicats sont simplement des fonctions qui renvoient une proposition (`Prop`). Par exemple, un prédicat unaire `P` qui a pour domaine l'ensemble des éléments de type `T` a le type `T -> Prop`.

```
Section Lesson3.

Variable T : Type.
Variable P Q : T -> Prop.
Variable R : T -> T -> Prop.
```

On se souvient qu'une fonction avec une arité $n > 1$ peut être représentée en λ-calcul par une fonction d'*ordre supérieur*, qui consomme le premier argument et retourne une fonction d'arité n pour les arguments suivants [1]. Le même principe s'applique donc aux *relations* (prédicats n-aires), comme illustré ci-dessus par la relation binaire `R`.

1 Le Quantificateur Universel (Pour Tout : $\forall$)

En logique, affirmer "Pour tout individu x, $P(x)$ est vrai" signifie que si on me donne n'importe quel individu x, je peux fournir une preuve de $P(x)$.

Informatiquement (grâce à Curry-Howard), c'est une FONCTION ! Une fonction qui prend en argument un élément `x` de type `T`, et qui renvoie une preuve du type `P x`. C'est ce qu'on appelle un **type dépendant** (le type de retour dépend de la valeur de l'argument).

Ainsi, l'implication simple $A \to B$ que nous connaissons bien n'est en fait qu'un $\forall$ dégénéré où le type de retour ne dépend pas de l'argument ! C'est pour cela qu'on utilise la même tactique `intro` / `intros`.

$$\frac{\Gamma, x : T \vdash M : P(x)}{\Gamma \vdash \lambda x.M : \forall x : T.P(x)} \forall I$$

```
Lemma forall_intro : (forall x : T, P x -> Q x) -> (forall x : T, P x)
   -> (forall y : T, Q y).
Proof.
  intros H_impl H_P.
```

1. Cette technique permettant de passer de l'ensemble/type des fonctions $A \times B \to C$ au type $A \to (B \to C)$ est souvent appelée "curryfication", en référence au logicien Haskell Curry qui l'utilisait extensivement. Il a aussi donné son nom au populaire langage de programmation fonctionnelle `Haskell`.

Notre but commence par `forall y : T, ...`. Nous voulons prouver que la propriété est vraie pour un `y` arbitraire. Nous l'introduisons dans le contexte.

```
  intro y.
```

Maintenant que nous avons un `y` spécifique (mais quelconque), notre but est `Q y`.

```
  apply H_impl.
```

Remarque 1: L'Unification Magique

Comment Rocq a-t-il su utiliser `H_impl` ? Notre hypothèse `H_impl` est une démonstration abstraite valable pour *n'importe quel* individu x. Or notre but était de prouver `Q y`. Sous le capot, Rocq a "superposé" la conclusion de l'hypothèse (`Q x`) avec notre but actuel (`Q y`). Il a déduit de lui-même qu'il fallait instancier l'inconnue abstraite x par l'individu concret y pour que la superposition soit parfaite !
Ce processus purement mécanique de mise en correspondance de deux expressions (ou arbres syntaxiques) pour déterminer l'identité des variables s'appelle l'**unification**. Il libère l'utilisateur de l'obligation de préciser systématiquement l'individu pour lequel il applique un théorème universel.

```
  apply H_P.
Qed.
```

2 Le Quantificateur Existentiel (Il Existe : $\exists$)

Du point de vue constructiviste, prouver "Il existe un x tel que $P(x)$", ce n'est pas simplement dire "ça doit bien exister quelque part". Il faut EN EXHIBER UN !

Une preuve d'existence est donc une PAIRE constituée de :

1. Un "témoin" x de type T (la valeur concrète).
2. Une preuve que ce témoin x satisfait bien la propriété P.

Pour construire une telle paire, on utilise la tactique `exists` en lui fournissant notre témoin.

$$\frac{\Gamma \vdash t : T \qquad \Gamma \vdash M : P(t)}{\Gamma \vdash (t, M) : \exists x : T, P(x)} \exists I$$

```
Variable t_defaut : T.

Lemma exists_intro : (forall x : T, P x) -> (exists y : T, P y \/ Q y).
Proof.
  intros H_P.
```

Nous utilisons notre élément `t_defaut` (qu'on a supposé par avance exister dans notre contexte pour éviter l'échec sur les domaines vides) comme témoin de l'existence.

```
  exists t_defaut.
```

Remarquez que le but a changé : Rocq a remplacé tous les `y` par `t_defaut`. Il ne nous reste plus qu'à en donner la preuve.

```
  left.
  apply H_P.
Qed.
```

À l'inverse, si nous possèdons une hypothèse existentielle $\exists x, P(x)$, nous voulons en extraire le témoin et la preuve associée. Comme c'est une paire sous le capot (tout comme le ET), on utilise la tactique `destruct`.

```
Lemma exists_elim : (exists x : T, P x /\ Q x) -> (exists y : T, P y).
Proof.
  intros H_exists.
```

On détruit l'hypothèse pour récupérer le témoin (que l'on nomme `x0`) et la preuve qu'il satisfait la propriété (qu'on nommera `H_PQ`).

```
  destruct H_exists as [x0 H_PQ].
  destruct H_PQ as [H_P H_Q].
  exists x0.
  exact H_P.
Qed.
```

3 L'Égalité Propositionnelle (=)

En mathématiques, l'égalité est la notion la plus fondamentale. Dans Rocq, `x = y` est une proposition. Dire que `x = y` est prouvable, c'est dire que x et y dénotent formellement le même objet.

L'outil principal pour manipuler l'égalité est la RÈGLE DE SUBSTITUTION : Si $x = y$, alors tout ce qui est vrai pour x est vrai pour y. C'est la tactique `rewrite`.

Remarque 2: Le Principe de Leibniz

Ce comportement trouve ses racines philosophiques dans le **principe d'identité des indiscernables** de Leibniz (la substitution des identiques *salva veritate*), dont nous reparlerons plus longuement la semaine prochaine au moment d'aborder la définition sous-jacente de l'égalité dans Rocq.

```
Lemma egalite_sym : forall x y : T, x = y -> y = x.
Proof.
  intros x y H_eq.
```

Nous savons que x et y sont égaux (hypothèse `H_eq`). Nous voulons prouver $y = x$. La tactique `rewrite` permet de remplacer le membre gauche de l'égalité par le membre droit partout dans le but.

```
  rewrite H_eq.
```

Notre but s'est transformé en $y = y$. La tactique `reflexivity` prouve instantanément que tout terme est égal à lui-même.

```
  reflexivity.
Qed.
```

4 Égalité Définitionnelle vs Propositionnelle

Rocq distingue deux types d'égalités :

1. **L'égalité PROPOSITIONNELLE** ($x = y$) : c'est un énoncé logique qu'il faut prouver (par exemple avec `rewrite`, en appliquant des lemmes).

2. **L'égalité DÉFINITIONNELLE** ($\equiv$) : c'est le fait que deux expressions se "calculent" pour donner le même résultat de manière purement mécanique (via $\beta\eta$-équivalence et déroulage de définitions).

La magie, c'est que la tactique `reflexivity` réussit *modulo* l'égalité définitionnelle ! Le système évalue discrètement les termes.

```
Definition doubler (n : nat) : nat := n + n.

Lemma reflexivite_puissante : doubler 2 = 4.
Proof.
```

Pour un logicien minutieux sur papier, il y a un petit travail de démonstration ici (par exemple avec les axiomes de Peano). Mais pour Rocq, `doubler 2` s'évalue mécaniquement en $2+2$, puis en 4. Ensuite il constate que $4 = 4$ est correct et conclut.

```
  reflexivity.
Qed.
```

Remarque 3: Calcul Automatique

C'est cette intégration du calcul AU CŒUR MÊME des règles de déduction qui rend la théorie des types intuitionniste particulièrement adéquate pour la formalisation assistée par ordinateur, contrairement à des fondations plus classiques tels que la théorie des ensembles ZFC.

5 Exercices à faire

Exercice 1: Transitivité de l'égalité

Prouvez formellement que l'égalité propositionnelle est transitive.

```
Lemma egalite_trans : forall x y z : T, x = y -> y = z -> x = z.
Proof.
  (* ECRIVEZ VOTRE PREUVE ICI *)
Admitted.
```

Exercice 2: Raisonnement Équationnel Pratique

Montrez comment utiliser `rewrite` pour substituer y par z au sein du prédicat binaire R.

```
Lemma equation_exemple : forall x y z : T,
  R x y -> y = z -> R x z.
Proof.
  (* ECRIVEZ VOTRE PREUVE ICI *)
Admitted.
```

```
End Lesson3.
```