

Fondements des systèmes de preuves – TP n° 3

## Réécritures et codages au second ordre

**Exercice 1 (Réécriture)** On considère en Coq les deux définitions suivantes :

```
Definition pred (n : nat) := match n with 0 => 0 | S p => p end.
Definition null (n : nat) := match n with 0 => True | S _ => False end.
```

1. En utilisant la fonction `pred`, montrer que : `forall x y : nat, S x = S y -> x = y`
2. En utilisant la fonction `null`, montrer que : `forall x : nat, S x <> 0`

On définit à présent le prédicat de parité `Even : nat -> Prop` et la fonction de test correspondante `even : nat -> bool` en posant :

```
Definition Even (n : nat) := exists p : nat, n = 2 * p.

Fixpoint even (n : nat) : bool :=
  match n with
  | 0   => true
  | S p => negb (even p)          (* Nécessite un "Require Import Bool." *)
  end.                          (* pour accéder à la constante "negb". *)
```

3. Tester cette fonction à l'aide de la commande `Eval compute in ...`
4. Montrer que : `forall n : nat, Even n <-> even n = true`

**Exercice 2 (L'égalité de Leibniz)** En théorie des types simples intuitionniste, on *définit* l'égalité de Leibniz (entre deux objets  $x$  et  $y$  d'un même type simple  $\tau$ ) à partir de l'implication et de la quantification universelle en posant :

$$x =_{\tau} y \quad \equiv \quad \forall P^{\tau \rightarrow o} (P x \Rightarrow P y).$$

(On notera l'utilisation d'une implication et non d'une équivalence logique.) Dans cet exercice, on se fixe un type `T : Set` en paramètre à l'aide de la commande `Parameter T : Set.`

1. Définir une constante `EQ : T -> T -> Prop` représentant cette définition de l'égalité.
2. Montrer que cette relation est réflexive et transitive.
3. Montrer qu'elle est symétrique.
4. Démontrer l'équivalence : `forall x y : T, EQ x y <-> x = y`

**Exercice 3 (Conjonction et disjonction)** En théorie des types simples intuitionniste, on peut aussi définir la conjonction et la disjonction à partir de  $\Rightarrow$  et  $\forall$  en posant :

$$A \wedge B \quad \equiv \quad \forall^o X ((A \Rightarrow B \Rightarrow X) \Rightarrow X)$$

$$A \vee B \quad \equiv \quad \forall^o X ((A \Rightarrow X) \Rightarrow (B \Rightarrow X) \Rightarrow X)$$

1. Définir en Coq une constante `ET` : `Prop -> Prop -> Prop` représentant la conjonction définie par le codage ci-dessus.
2. Montrez que ce connecteur satisfait les règles d'introduction et d'élimination attendues :
  - `forall A B : Prop, A -> B -> ET A B`
  - `forall A B : Prop, ET A B -> A` (de même pour la propriété symétrique)
3. En déduire que les conjonctions `ET` et `&` sont équivalentes :
$$\text{forall } A B : \text{Prop}, \text{ET } A B \leftrightarrow A \wedge B$$
4. Même exercice avec le connecteur `OU`.
5. En suivant l'esprit des codages ci-dessus, proposer un codage pour les constantes propositionnelles `⊥` (absurdité) et `⊤` (vérité), et vérifiez les équivalences avec `False` et `True`.

**Exercice 4 (Quantification existentielle)** Dans le même esprit, la quantification existentielle sur un type  $\tau$  est définissable en termes de  $\Rightarrow$  et  $\forall$  en posant<sup>1</sup> :

$$\exists^\tau x P(x) \equiv \forall X^o (\forall^\tau x (P(x) \Rightarrow X) \Rightarrow X)$$

Comme dans l'exercice 2, on se donne en paramètre un type de données `T` : `Set`.

1. Définir en Coq une constante `EX` : `(T -> Prop) -> Prop` représentant le quantificateur existentiel défini ci-dessus.
2. Montrez que ce quantificateur satisfait les règles d'introduction et d'élimination de la quantification existentielle. Comment s'expriment-elles en Coq ?
3. Montrez que les quantificateurs `EX` et `exists` sont équivalents en Coq :

$$\text{forall } P : T \rightarrow \text{Prop}, \quad \text{EX } P \leftrightarrow \text{exists } x : T, P \ x$$

**Exercice 5 (Le principe de récurrence)** On considère trois constantes `T` : `Set`, `o` : `T` et `s` : `T -> T` données en paramètre. La plus petite sous-classe de `T` contenant `o` et close par la fonction `s` est représentée en Coq par le prédicat `N` : `T -> Prop` défini par :

**Definition** `N` (`x` : `T`) :=  
`forall P : T -> Prop, P o -> (forall y, P y -> P (s y)) -> P x`

1. Montrez que : `N o`.
2. Montrez que : `forall x : T, N x -> N (s x)`.

Une fois que cette classe est définie, il est possible de restreindre les quantifications universelle et existentielle aux éléments de cette classe en utilisant les constructions

$$\text{forall } x : T, N \ x \rightarrow A \ x \quad \text{et} \quad \text{exists } x : T, N \ x \wedge A \ x.$$

3. Montrez que dans la classe définie par `N`, le principe de récurrence est satisfait :

$$\text{forall } A : T \rightarrow \text{Prop}, \\ A \ o \rightarrow (\text{forall } x, N \ x \rightarrow A \ x \rightarrow A \ (s \ x)) \rightarrow \text{forall } x, N \ x \rightarrow A \ x.$$

---

<sup>1</sup>Dans l'égalité ci-dessus, on suit les conventions de la logique moderne, où les quantificateurs ont une précedence plus grande que les connecteurs logiques. La formule  $\forall x (P(x) \Rightarrow X) \Rightarrow X$  se lit donc  $[\forall^\tau x (P(x) \Rightarrow X)] \Rightarrow X$ . Cependant, en Coq on utilise la convention inverse (comme dans la plupart des mathématiques informelles). Attention au parenthésage au moment de la formalisation !