Fondements des systèmes de preuves – TP n° 7

Système F et polymorphisme

Exercice 0 (Les sortes `Prop` et `Set`)

1. Vérifiez (à l'aide de la commande `Check`) les sortes des types suivants :
   - `A -> A` (où `A : Prop` est déclaré en paramètre)
   - `forall A : Prop, A -> A`
2. Meme question en remplaçant `A : Prop` par `A : Set`. Que constate-t-on ?

En Coq, la sorte `Prop` est imprédicative ; il est donc possible de représenter dans cette sorte tous les types du système F (ainsi que les termes correspondants) en suivant la correspondance syntaxique ci-dessous :[1]

$A \to B$	$\equiv$	`A -> B`	$\forall X \ldots$	$\equiv$	`forall X : Prop, ...`
$\lambda x : A \ldots$	$\equiv$	`fun (x : A) => ...`	$\Lambda X \ldots$	$\equiv$	`fun (X : Prop) => ...`

Exercice 1 (Les booléens) Les booléens du système F sont définis par :

```
Definition BOOL          :=  forall X : Prop, X -> X -> X.
Definition TRUE : BOOL   :=  fun (X : Prop) (x y : X) => x.
Definition FALSE : BOOL  :=  fun (X : Prop) (x y : X) => y.
```

1. Écrire la construction "if ... then ... else ..." sous la forme d'une fonction
   `IF : forall X : Prop, BOOL -> X -> X -> X.`
2. Écrire la fonction booléenne de négation.
3. Écrire les fonctions booléennes « et » et « ou ».

Exercice 2 (Le produit cartésien) Le produit cartésien du système F est défini par :

```
Definition PROD (A B : Prop) := forall (X : Prop), (A -> B -> X) -> X.
```

1. Définir la fonction de construction de paire :
   `PAIR : forall A B : Prop, A -> B -> PROD A B.`
2. Définir les fonctions de projection :
   `FST : forall A B : Prop, PROD A B -> A` et
   `SND : forall A B : Prop, PROD A B -> B.`
3. Quelles sont les égalités définitionnelles attendues ? Vérifiez-les.
4. N'avez-vous pas déjà fait le même exercice dans un autre contexte ?

[1] Il est possible de faire les mêmes constructions dans la sorte `Set` à condition de lancer le système avec l'option `-set-impredicative`, qui rend la sorte `Set` imprédicative. Pendant longtemps, cela a été le comportement par défaut en Coq, avant qu'on ne découvre que l'imprédicativité de `Set` est incompatible avec la logique classique.

Exercice 3 (Les entiers) Les entiers du système F sont définis par :

```
Definition NAT         :=  forall X : Prop, X -> (X -> X) -> X.
Definition ZERO : NAT  :=  fun (X : Prop) (x : X) (f : X -> X) => x.
Definition ONE  : NAT  :=  fun (X : Prop) (x : X) (f : X -> X) => f x.
Definition TWO  : NAT  :=  fun (X : Prop) (x : X) (f : X -> X) => f (f x).
```

1. Écrire la fonction successeur `SUCC : NAT -> NAT`.
2. Écrire l'addition (`PLUS`) et la multiplication (`MULT`).
3. Écrire la fonction d'Ackermann `ACK`, définie par :

$$\begin{aligned} \mathrm{ack}(0,\ m) &= m+1 \\ \mathrm{ack}(n+1,\ 0) &= \mathrm{ack}(n,\ 1) \\ \mathrm{ack}(n+1,\ m+1) &= \mathrm{ack}(n,\ \mathrm{ack}(n+1,\ m)) \end{aligned}$$

Exercice 4 (La fonction prédécesseur)

1. Écrire une fonction `STEP : PROD NAT NAT -> PROD NAT NAT` qui envoie le couple (n, m) sur le couple $(n+1, n)$.
2. En déduire une implémentation de la fonction prédécesseur dans le système F.
3. En utilisant une astuce similaire, écrire dans le système F la fonction qui effectue la division euclidienne par 2.

Exercice 5 (Listes & C$^{\text{ie}}$) Le type des listes dans le système F est défini par

```
Definition LIST (A : Prop) := forall (X : Prop), X -> (A -> X -> X) -> X.
```

1. Définir les fonctions de construction :
   `NIL  : forall A : Prop, LIST A` et
   `CONS : forall A : Prop, A -> LIST A -> LIST A`.
2. Écrire les fonctions "usuelles" :
   `LENGTH  : forall A : Prop, LIST A -> NAT`,
   `REVERSE : forall A : Prop, LIST A -> LIST A`,
   `MAP     : forall A B : Prop, (A -> B) -> LIST A -> LIST B`.
3. ($\star\star$) Implémenter la fonction de tri fusion :
   `MERGE_SORT : forall A : Prop, (A -> B -> BOOL) -> LIST A -> LIST A`.
4. Définir un type `BTREE A` des arbres binaires à étiquettes dans `A` (sur les nœuds internes).
5. ($\star\star\star$) Implémenter la fonction de tri par tas :
   `HEAP_SORT : forall A : Prop, (A -> B -> BOOL) -> LIST A -> LIST A`.