Assistants de preuve

TD 1- Du Calcul des Constructions au Calcul des Constructions Inductives

1 Types inductifs de base

1.1 Booléens

On se donne

```
Inductive bool : Type := true : bool | false : bool.
```

1- Définir la négation booléenne *negb* et la conjonction booléenne *andb* dans le CCI (sur papier ou dans Coq).

2- Expliciter les étapes de la normalisation des expression $\lambda x : \texttt{bool}.\, negb\ (andb\ false\ x)$ et $\lambda x : \texttt{bool}.\, negb\ (andb\ x\ false)$ (dans Coq, on pourra utiliser la commande `Eval`). Que remarquez-vous ?

1.2 Disjonction

On se donne, dans Coq, la définition de schéma de type suivant:

```
Inductive sum (A B:Type) : Type :=
| inl : A -> sum A B
| inr : B -> sum A B.
```

1- Donner une expression équivalente dans la syntaxe de Objective Caml. Comment s'écrit l'opérateur d'analyse de cas (l'élimination) de `sum` en Objective Caml ?
2- Donner une expression équivalente dans le système F_ω, sous la forme

- d'un type $\texttt{sum}' : \texttt{Prop} \to \texttt{Prop} \to \texttt{Prop}$,
- de fonctions $\texttt{inl}' : \forall A\, B : \texttt{Prop}, A \to \texttt{sum}'\ A\ B$ et $\texttt{inr}' : \forall A\, B : \texttt{Prop}, B \to \texttt{sum}'\ A\ B$,
- d'une fonction d'analyse de cas (non dépendant) $case_{\texttt{sum}'} : \forall A\, B\, P : \texttt{Prop}, (A \to P) \to (B \to P) \to sum\ A\ B \to P$.

3- Définir dans le Calcul des Constructions Inductives la fonction de type $\forall A, B : Type.\texttt{sum}\ A\ B \to \texttt{bool}$ qui indique quelle composante de la somme est choisie.
4- Avec la commande `Extraction` *ident* de Coq vers Objective Caml, vérifier la correspondance entre les définitions Coq et les définitions correspondantes en Objective Caml.

1.3 Conjonction

Donner une définition inductive dans `Prop` pour la conjonction $\wedge$. Donner un terme de preuve prouvant $\forall A, B : Prop. A \wedge B \rightarrow B \wedge A$ où $\wedge$ représente cette conjonction.

2 Expressivité : les entiers de Church

Un entier de Church est un entier représenté par une fonctionnelle de la forme

$$\lambda x.\lambda f.f(\ldots(f(x))\ldots).$$

On se place dans le calcul des constructions (les sortes sont appelées `Prop` et `Type` et le produit est noté avec $\forall$). Si A est un type, on définit l'égalité sur A par

$$(x =_A y) \triangleq \forall P : (A \rightarrow Prop)\ P(x) \rightarrow P(y)$$

et la négation $\neg A$ de A par

$$\neg A \triangleq A \rightarrow \forall C.\, C\,.$$

A- Définition des entiers dans la couche `Prop` du Calcul des Constructions

Dans la suite s représente la sorte `Prop`.

1. Donner une expression close N de type s exprimant le type des entiers de Church. Comment s'expriment alors 0 et l'opération « successeur » (notée S) ?
2. Comment définir l'addition et la multiplication sur N ?
3. Peut-on définir le prédécesseur sur N ?
4. On pose $IND(n) \triangleq \forall P : N \rightarrow \texttt{Prop}.(P(0) \rightarrow (\forall m : N.\, P(m) \rightarrow P(S(m))) \rightarrow P(n)$. Peut-on dériver le principe de récurrence $\forall n : N.\, IND(n)$?
5. Peut-on exprimer (dans `Prop`) l'énoncé $\forall n, m : N, S(n) = S(m) \rightarrow n = m$? Si oui, peut-on le prouver ? Si non, peut-on prouver $\forall n, m : N, IND(n) \rightarrow IND(m) \rightarrow S(n) = S(m) \rightarrow n = m$?
6. Peut-on exprimer (dans `Prop`) l'énoncé $\forall n : N, S(n) \neq 0$? Si oui, peut-on le prouver ? Si non, peut-on prouver $\forall n : N, S(n) \neq 0$?

B- Mêmes questions en prenant pour s la sorte `Type` du Calcul des Constructions

C- Mêmes questions en restant dans `Type` mais en se restreignant au système F_ω.

D- Mêmes questions en se plaçant dans $F_{\omega.2}$.

Remarque: on pourra (ou non) donner un type polymorphe aux entiers de Church; en ce cas, la représentation des entiers pourra expliciter une abstraction de types (comme dans $\lambda X.\lambda x : X.\lambda f : X \rightarrow X.f(f(f(x))))$.
Remarque: les démonstrations de non-prouvabilité sont complexes. On pourra déjà s'intéresser à l'interprétation "proof-irrelevant" du Calcul des Constructions. Elle consiste à trivialiser les dépendances de preuves et à interpréter `Prop` comme un ensemble à deux valeurs.