Assistants de preuve

TD 2- Les spécificités du Calcul des Constructions Inductives

1 Prédicats inductifs

A- Donner une définition inductive `pair : nat -> Prop` du prédicat « être pair ».
B- Caractériser par un prédicat inductif `exp : nat -> nat -> nat -> Prop` à deux constructeurs le graphe de la fonction $n^p = q$ sur les entiers naturels.

2 Types récursifs

A- Donner en Coq une définition inductive avec paramètres qui caractérise le type suivant des listes polymorphes en ML

```
type 'a list = nil | cons of 'a * 'a list
```

B- On définit respectivement le produit, le type singleton et les entiers naturels par

```
Inductive prod (A B : Type) : Type := pair : A -> B -> prod A B.
Inductive unit : Type := tt : unit.
Inductive nat : Type := O : nat | S : nat -> nat.
```

Donner une expression `prodn` du CCI de type `Type` $\rightarrow$ `nat` $\rightarrow$ `Type` qui construise le produit n-aire d'un type A donné (c.-à-d. `prodn` A n est $A \times \ldots \times A$ (n fois)).

Donner une expression `length` de type $\forall A.$ `list` $A \rightarrow$ `nat` qui calcule la longueur d'une liste.

Donner une expression `embed` de type $\forall A. \forall l :$ `list` $A.$ `prodn` A (`length` l) qui traduise une liste en un n-uplet.

C- Définir un récurseur de type $\forall A :$ `Type`$. \forall C :$ `Type`$. C \rightarrow (A \rightarrow$ `list` $A \rightarrow C \rightarrow C) \rightarrow$ `list` $A \rightarrow C$.

Modifier ce récurseur pour qu'il puisse construire des fonctions à codomaine dépendant sur les listes (c'est-à-dire des fonctions dans un type produit $\Pi l :$ `list` $A.C(l)$).

3 Décroissance des points-fixes

A- Le point-fixe suivant est-il bien fondé pour le CCI et pourquoi ?

```
Fixpoint leq (n p: nat) {struct n} : bool :=
  match n with
  | O => true
  | S n' => match p with O => false | S p' => leq n' p' end
  end.
```

B- Le point-fixe suivant est-il bien fondé pour le CCI et pourquoi ?

```
Definition exp (p:nat) :=
 (fix f (n:nat) : nat :=
  match leq n p with
  | true => S 0
  | false => f (S n) + f (S n)
  end)   0.
```

C- Le point-fixe suivant est-il bien fondé pour le CCI et pourquoi ?

```
Definition ackermann := fix f (n:nat) : nat -> nat := match n with
  | 0 => S
  | S n' => fix g (m:nat) : nat := match m with
                                   | 0 => f n' (S 0)
                                   | S m' => f n' (g m')
                                   end
  end.
```

Accessoirement, que valent `ackermann` 2 3 et `ackermann` 3 2 ?

4 Élimination forte

Soit t_1 et t_2 des termes arbitraires de types T_1 et T_2 dans un contexte donné. La fonction

```
Definition g (b:bool) := match b with true => t1 | false => t2 end.
```

est-elle typable ? Si oui, donner la clause `return` correspondante.

5 Restrictions d'élimination liées aux sortes

A- On pose

```
Inductive True : Prop := I : True.
```

Exhiber une fonction de `unit` vers `True` dont on peut montrer qu'elle est une bijection.

B- On pose

```
Inductive BOOL : Prop := TRUE : BOOL | FALSE : BOOL.
```

Peut-on montrer l'équivalence de `bool` et `BOOL` ? Montrer que si c'était le cas, on pourrait nier le principe d'indiscernabilité des preuves (« proof-irrelevance », c'est-à-dire $\forall P : Prop\, \forall p\, q : P.\, p = q$) dans `Prop`.