1 Injection and discrimination

Consider the type of lists, defined in the prelude of Coq:

```coq
Inductive list (A:Type) :=
  nil | cons (_:A) (_:list A).
```

Prove, without using the dedicated tactics `injection` and `discriminate`, the following properties on lists:

1. Injectivity of `cons`: \( \text{cons } x_1 l_1 = \text{cons } x_2 l_2 \rightarrow x_1 = x_2 \land l_1 = l_2 \)
2. Discrimination of `nil` and `cons`: \( \text{nil} \not= \text{cons } x \)

2 Strict positivity

2.1 Encoding pure \(\lambda\)-calculus in a non-positive inductive type

In this section we analyze the consequence of having an inductive type which does not comply with the strict positivity condition. We (partially) simulate the inductive definition

```coq
Inductive lambda := Lam (_:lambda->lambda).
```

by assuming the introduction and case-analysis rules:

```coq
Parameter lambda : Type.
Parameter Lam : (lambda->lambda) -> lambda.
Parameter match_lambda : forall P:lambda -> Type, <todo> -> forall l, P l.
```

The \(\iota\)-reduction is represented by an equation:

```coq
Parameter lambda_eq : forall P H f, match_lambda P H (Lam f) = H f.
```

1- Encode the pure \(\lambda\)-calculus in this type, by defining application `app : lambda -> lambda -> lambda` and the \(\beta\)-equality `app (Lam f)x = f x`
2- Prove that any function `f : lambda -> lambda` has a fixpoint. That is, there exists a term `t` such that `f t = t`.
3- Show that the above axiomatization of `lambda` does not introduce an inconsistency by exhibiting a model. Complete the following piece of theory:

```coq
Definition L : Type := <todo>.
Definition Li (f:L->L) : L := <todo>.
Definition Lm (P:L->Type) (H:<todo>) (l:L) : P l := <todo>.
Lemma L_eq P H f : Lm P H (Li f) = H f. <todo>.
```

Now, consider the recursive scheme for `lambda` considering that each image of `f` is structurally smaller than `Lam f`.

4- Introduce a parameter `rec.lambda` encoding this recursive scheme.
5- Show that it is inconsistent.
6- Explain why the following fixpoint is not accepted ?
Inductive id := Id (_:forall A:Prop,A->A).

Fixpoint F (i:id) :=
  match i with
  | Id f => F (f id i)
  end.

3 Termination of fixpoints

Are the following fixpoints well-formed ? explain why ?

Fixpoint leq (n p: nat) {struct n} : bool :=
  match n, p with
  | O, _ => true
  | S _, 0 => false
  | S n', S p' => leq n' p'
  end.

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

Definition ackermann1 := fix f (n m:nat) : nat :=
  match n, m with
  | O, _ => S m
  | S n', 0 => f n' (S O)
  | S n', S m' => f n' (f n m')
  end.

Definition ackermann2 := fix f (n:nat) : nat -> nat :=
  match n with
  | O => S
  | S n' => fix g (m:nat) : nat :=
  match m with
  | O => f n' (S O)
  | S m' => f n' (g m')
  end
  end.

4 The type $W$ of well-founded trees

The type $W$ of well-founded trees is parameterised by a type $A$ and a family of types $B:A->Type$. It has only one constructor and is defined by :

Inductive W (A:Type) (B:A -> Type) : Type :=
  node : forall (a:A), (B a -> W A B) -> W A B.

The type $A$ is used to parameterised the nodes and the type $B a$ give the arity of the node parameterised by $a$.

1. Give the type of dependent elimination for type $W$ on sort Type.

2. In order to encode the type $nat$ of natural numbers with $O$ and $S$, we need two types of nodes. We take $A = bool$. The constructor $O$ corresponds to $a = false$, it does not expect any argument so we take $B false=empty$. The constructor $S$ corresponds to $a = true$, it takes one argument, we define $B true=unit$. Using this encoding, give the terms corresponding to $nat$, $O$ et $S$.

3. Propose an encoding using $W$ for the type $tree$ of binary trees parameterised by a type of values $V$, which means that we have a constructor $leaf$ of type $tree V$ and a constructor $bin$ of type $tree V -> V -> tree V -> tree V$. Define the type and its constructors using this encoding.
4. Given a variable \( n \) of type \( \text{nat} \), build two functions \( f_1 \) and \( f_2 \) of type \( \text{unit} \rightarrow \text{nat} \) such that \( \forall x: \text{unit}, f_i \ x = n \) is provable but such that \( f_1 \) and \( f_2 \) are not convertible.

5. Which consequence does it have on the encoding of \( \text{nat} \) using \( W \)?