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-

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

Definition andb (b b':bool) := match b with true => b' | false => false end.
```

2-

$$\lambda x : bool.\, negb\ (andb\ false\ x)$$
$$\rightarrow_\iota \lambda x : bool.\, negb\ (false)$$
$$\rightarrow_\iota \lambda x : bool.\, false$$

Mais $\lambda x : bool.\, negb\ (andb\ x\ false)$ ne se ι-réduit pas : la réduction n'est pas symétrique et privilégie un « indice de séquentialité » dans la définition de *andb*.

1.2 Disjonction

1-

```
type ('a,'b) sum = Inl of 'a | Inr of 'b
let case (x : ('a,'b) sum) fa fb = match x with Inl a -> fa a | Inr b -> fb b
```

2-

```
Definition sum' (A B:Prop) := forall P:Prop, (A -> P) -> (B -> P) -> P.
Definition inl' (A B:Prop) (H:A) : sum' A B := fun (P:Prop) (H1:A -> P) (H2:B -> P) => H
Definition inr' (A B:Prop) (H:B) : sum' A B := fun (P:Prop) (H1:A -> P) (H2:B -> P) => H
Definition case_sum' (A B P:Prop) (H1:A -> P) (H2:B -> P) (H:sum' A B) :=
  H P H1 H2.
```

3-

```
Definition select (A B:Type) :=
  fun (b:sum A B) => match b with inl _ => true | inr _ => false end.
```

1.3 Conjonction

```
Inductive and (A B:Prop) : Prop := conj : A -> B -> and A B.

Theorem and_comm : forall A B : Prop, and A B -> and B A.
exact (fun A B H => match H with conj a b => conj _ _ b a end).
```

2 Expressivité : les entiers de Church

A- Formalisé en Coq, on obtient:

```
(**********************************)
(* Entiers au niveau Prop dans CC *)

Definition eq := fun (A : Prop) (a b:A) => forall P:A->Prop, P a -> P b.
Definition neg := fun (A : Prop) => A -> forall C : Prop, C.

Definition N := forall C:Prop, C -> (C -> C) -> C.
Definition O : N := fun C x f => x.
Definition S : N -> N := fun n => fun C x f => f (n C x f).
Definition plus : N -> N -> N := fun n m => fun C x f => n C (m C x f) f.
Definition mult : N -> N -> N := fun n m => fun C x f => n C x (fun x => m C x f).

Definition prod := fun A B : Prop => forall C:Prop, (A -> B -> C) -> C.
Definition pair := fun A B : Prop => fun a b => fun C (f:A->B->C) => f a b.
Definition fst := fun A B : Prop => fun p : prod A B => p A (fun x _ => x).
Definition snd := fun A B : Prop => fun p : prod A B => p B (fun _ y => y).

Definition pred : N -> N := fun n =>
  fst _ _ (n (prod N N) (pair _ _ O O)
    (fun p => pair _ _ (snd _ _ p) (S (snd _ _ p)))).

Definition IND := fun n:N =>
  forall P:N->Prop, P O -> (forall n, P n -> P (S n)) -> P n.

(* Non prouvable : en considérant le modèle booléen (mais non
   proof-irrelevant) interprétant Prop comme l'ensemble à deux types,
   soit le type vide, soit le type de tous les lambda-termes purs,
   alors, l'interprétation de N (qui est habité) est l'ensemble de
   tous les lambda-termes, alors que l'interprétation de "IND n" est
   une proposition qui est habitée ssi n est un lambda-terme construit
   par itération du lambda-terme S au dessus du lambda-terme 0. "IND
   n" ne recouvre donc pas l'ensemble de tous les lambda-termes (il ne
```

```
   recouvre que le sous-ensemble des entiers de Church) et "forall
   n:N, IND n" est faux (càd vide) dans le modèle. [référence: Herman
   Geuvers, Induction Is Not Derivable in Second Order Dependent Type
   Theory, TLCA 2001, en y étendant la construction au cas du CC comme
   dans Stefanova-Geuvers, A Simple Model Construction for the
   Calculus of Constructions, TYPES 1995]

Theorem ind : forall n:N, IND n.
*)

(* A priori non prouvable : peut-on construire un modèle de termes qui
   le contredise ? (le point-clé qui nécessite l'induction est le lemme
   "pred (S n) = n"; il nécessite l'induction parce que le prédécesseur
   se reconstruit par itération à partir de zéro)

Theorem inj_S : forall n m : N, eq N (S n) (S m) -> eq N n m.
*)

Theorem inj_S : forall n m : N, IND n -> IND m -> eq N (S n) (S m) -> eq N n m.
Proof.
intros n m Hn Hm H.
assert (H':forall n : N, IND n -> eq _ (pred (S n)) n).
 intros p Hp. apply Hp.
  exact (fun _ x => x).
  intros n0 Heqn0. pattern n0 at 2. apply Heqn0. exact (fun _ x => x).
apply (H' m Hm).
apply H.
pattern n at 1.
apply (H' n Hn).
exact (fun _ x => x).
Qed.

(* Non prouvable parce CC admet une interprétation "proof-irrelevant" qui
   égalise tout terme de type un type dans Prop (cf ci-dessous).

Theorem O_S : forall n : N, neg (eq _ O (S n)).
*)
```

Non prouvabilité de `O_S` via l'interprétation suivante (cf Thierry Coquand, Metamathematical Investigations of a Calculus of Constructions) :

$$
\begin{array}{lcll}
\phi(\texttt{Prop}) & = & \{true, false\} & \\
\phi(K \to L) & = & \phi K \to \phi(L) & \\
\phi(A \to L) & = & \phi L & \\
 & & & \\
\phi_\rho(X) & = & \rho(X) & \\
\phi_\rho(\forall X : K.\, A) & = & true & \text{si } \phi_{\rho,(X:=f)}(A) = true \text{ pour tout } f \in \phi(K) \\
 & = & false & \text{sinon} \\
\phi_\rho(A \to B) & = & true & \text{si } \phi_\rho(A) = false \text{ ou } \phi_\rho(B) = true \\
 & = & false & \text{sinon} \\
\phi_\rho(\lambda x : A.\, B) & = & \phi_\rho(B) & \\
\phi_\rho(M\ t) & = & \phi_\rho(M) & \\
\phi_\rho(\lambda X : K.\, B) & = & f \in \phi(K) \mapsto \phi_{\rho,(X:=f)}(B) & \\
\phi_\rho(M\ N) & = & \phi_\rho(M)\ \phi_\rho(N) &
\end{array}
$$

dans laquelle K, L parcourent les types dans `Type`, X, M, N parcourent les constructeurs de types, A, B parcourent les propositions et x, t, u parcourent les preuves (ce sont des propriétés décidables à dérivation donnée). On prouve ensuite que

$\vdash M : K$ entraîne $\phi_\emptyset(M) \in \phi_\emptyset(K)$
$\vdash t : A$ entraîne $\phi_\emptyset(A) = true$
$\vdash A =_\beta B$ entraîne $\phi_\emptyset(A) = \phi_\emptyset(B)$

Enfin, on a $\phi_\emptyset(\forall A : Prop.\, A) = false$ mais $\phi_\emptyset(\forall n : N.\, S(n) = 0) = true$, d'où $\not\vdash \forall n : N.\, S(n) \neq 0)$.

Remarque: cette interprétation est décidable car $\phi(K)$ est de cardinal fini.

B- Formalisé en Coq, on obtient:

```
(*****************************************)
(* Entiers au niveau Type dans CC et F_w *)

Definition eq := fun (A : Type) (a b:A) => forall P:A->Prop, P a -> P b.
Definition neg := fun (A : Prop) => A -> forall C : Prop, C.

Definition N := Prop -> (Prop -> Prop) -> Prop.
Definition O : N := fun x f => x.
Definition S : N -> N := fun n => fun x f => f (n x f).
Definition plus : N -> N -> N := fun n m => fun x f => n (m x f) f.
Definition mult : N -> N -> N := fun n m => fun x f => n x (fun x => m x f).

(* pas de prédécesseur (et pas de fonction puissance) *)
(* cf Schwichtenberg 1976 et Statman 1979 *)
```

```
Definition IND := fun n:N =>
  forall P:N->Prop, P O -> (forall n, P n -> P (S n)) -> P n.

(* pas prouvable car le contraire est prouvable: il y a dans N des
entiers non standard (par exemple tout ceux de la forme \_.\_.A avec A
proposition quelconque)

Theorem ind : forall n : N, IND n.
*)

Theorem not_ind : (forall n : N, IND n) -> forall C:Prop, C.
Proof.
intros H C.
pose (n_non_standard := (fun _ _ => forall C:Prop, C) : N).
pose (True:=C->C).
pose (P_identity := (fun n:N => n True (fun X => X))).
apply (H n_non_standard P_identity).
exact (fun x => x).
intros. assumption.
Qed.

(* pas prouvables ??
Theorem inj_S : forall n m : N, eq N (S n) (S m) -> eq N n m.
Theorem inj_S : forall n m : N, IND n -> IND m -> eq N (S n) (S m) -> eq N n m.
*)

Theorem O_S : forall n : N, neg (eq _ O (S n)).
Proof.
intros n H C.
change C with (S n (C -> C) (fun _ => C)) in |- *.
apply H. exact (fun x => x).
Qed.
```

C- Même réponse qu'en B. Notons que si les entiers étaient en revanche dans `Prop` dans F_ω, on ne pourrait pas exprimer l'égalité sur N (et en particulier pas les énoncés `inj_S` et `O_S`).

D- Formalisé en Coq, on obtient (cf, sur `http://coq.inria.fr/contribs-fra.html`, la contribution IZF de Miquel) :

```
(*********************************************)
(* Entiers au niveau Type_2 dans CC_w et F_w.2 *)

Definition eq := fun (A : Type) (a b:A) => forall P:A->Prop, P a -> P b.
```

```
Definition neg := fun (A : Prop) => A -> forall C : Prop, C.

Definition Type2 := Type.
Definition Type1 := Type : Type2.

Definition N : Type2 := forall C:Type1, C -> (C -> C) -> C.
Definition O : N := fun C x f => x.
Definition S : N -> N := fun n => fun C x f => f (n C x f).
Definition plus : N -> N -> N := fun n m => fun C x f => n C (m C x f) f.
Definition mult : N -> N -> N := fun n m => fun C x f => n C x (fun x => m C x f).

Definition prod := fun A : Type1 => (A -> A -> A) -> A.
Definition pair := fun A : Type1 => fun a b => fun (f:A->A->A) => f a b.
Definition fst := fun A : Type1 => fun p : prod A => p (fun x _ => x).
Definition snd := fun A : Type1 => fun p : prod A => p (fun _ y => y).

Definition pred : N -> N := fun n => fun C x f =>
  fst _ (n _ (pair _ x x)
    (fun p => pair _ (snd _ p) (f (snd _ p)))).

Definition IND := fun n:N =>
  forall P:N->Prop, P O -> (forall n, P n -> P (S n)) -> P n.

(* pas prouvable car le contraire est prouvable: il y a dans N des
entiers non standard (par exemple tout ceux de la forme \_.\_.A avec A
proposition quelconque) -- cf B-

Theorem ind : forall n : N, IND n.
*)

Theorem inj_S : forall n m : N, IND n -> IND m -> eq N (S n) (S m) -> eq N n m.
Proof.
intros n m Hn Hm H.
assert (H':forall n : N, IND n -> eq _ (pred (S n)) n).
 intros p Hp. apply Hp.
  exact (fun _ x => x).
  intros n0 Heqn0. pattern n0 at 2. apply Heqn0. exact (fun _ x => x).
apply (H' m Hm).
apply H.
pattern n at 1.
apply (H' n Hn).
exact (fun _ x => x).
```

```
Qed.

Theorem O_S : forall n : N, neg (eq _ O (S n)).
Proof.
intros n H.
apply (H (fun n => n _ (forall C:Prop, C -> C) (fun _ => forall C:Prop, C))).
exact (fun _ x => x).
Qed.
```