Proof assistants

TD 1- From the Calculus of Constructions to the Calculus of Inductive Constructions

1 Basic inductive definitions

1.1 Booleans

We define

\[
\text{Inductive bool : Type := true : bool | false : bool.}
\]

1- Define boolean negation \text{negb} and boolean conjunction \text{andb} in CCI (on paper or with Coq).

2- Detail the normalisation steps of expressions \(\lambda x : \text{bool}. \text{negb} (\text{andb} \ false \ x)\) et \(\lambda x : \text{bool}. \text{negb} (\text{andb} \ x \ false)\) (in Coq, one can use the command \text{Eval}). What is remarkable ?

1.2 Disjonction

We define, in Coq, the following type scheme:

\[
\text{Inductive sum (A B:Type) : Type :=}
\]

\[
| \text{inl : A -> sum A B}
\]

\[
| \text{inr : B -> sum A B.}
\]

1- Give an equivalent definition using Objective Caml. How the case analysis operator (elimination) for \text{sum} can be written in Objective Caml ?

2- Give an equivalent expression in system \(F_\omega\), as follows

- a type \text{sum'} : \text{Prop} \rightarrow \text{Prop} \rightarrow \text{Prop},
- two functions \text{inl'} : \forall A B : \text{Prop}, A \rightarrow \text{sum'} A B and \text{inr'} : \forall A B : \text{Prop}, B \rightarrow \text{sum'} A B,
- a (non dependent) case analysis function \text{case}_{\text{sum'}} : \forall A B P : \text{Prop}, (A \rightarrow P) \rightarrow (B \rightarrow P) \rightarrow \text{sum A B} \rightarrow P.

3- Define in the Calculus of Inductive Constructions a function with type \(\forall A, B : \text{Type}. \text{sum A B} \rightarrow \text{bool} \) which returns which component of the sum is fulfilled.

4- With the command \text{Extraction ident} from Coq to Objective Caml, check the correspondance between Coq definitions and the corresponding definitions in Objective Caml.

1.3 Conjunction

Give an inductive definition in \text{Prop} for conjunction \(\land\). Give a proof term for the property \(\forall A, B : \text{Prop}. A \land B \rightarrow B \land A\) where \(\land\) is your conjunction.
2 Expressivity : Church’s numbers

A Church number is a natural number represented by a functional term of the form

\[ \lambda x.\lambda f.(\ldots(f(x))\ldots). \]

We use the calculus of constructions (sorts are called Prop and Type and the product is written \( \forall \)). If \( A \) is a type, we define equality on \( A \) with

\[ (x =_A y) \triangleq \forall P : (A \to Prop), P(x) \to P(y) \]

and negation \( \neg A \) of \( A \) with

\[ \neg A \triangleq A \to \forall C.C. \]

A- Definition of Church’s numbers at the Prop level in the Calculus of Constructions

In the following \( s \) represents the sort Prop.

1. Give a closed expression \( N \) of type \( s \) for the type of Church numbers. Write terms for 0 and the operation « successor » (written \( S \)) ?

2. How to define addition and multiplication on \( N \) ?

3. Can we define the predecessor function on \( N \) ?

4. We define \( IND(n) \triangleq \forall P : N \to Prop.P(0) \to (\forall m : N. P(m) \to P(S(m))) \to P(n). \) Can we derive the induction principle \( \forall n : N. IND(n) ? \)

5. Can we express (in Prop) the property \( \forall n, m : N, S(n) = S(m) \to n = m \) ? If yes can we prove it ? If not, can we prove \( \forall n, m : N, IND(n) \to IND(m) \to S(n) = S(m) \to n = m \) ?

6. Can we express (in Prop) the property \( \forall n : N, S(n) \neq 0 \) ? If yes can we prove it ? If not, can we prove \( \forall n : N, IND(n) \to S(n) \neq 0 \) ?

B- Same questions but with \( s \) the sort Type of the Calculus of Constructions

C- Same questions but with \( s \) the sort Type and if we are only in \( F_\omega \).

D- Same questions but with \( s \) the sort Type and if we are only in \( F_\omega^2 \).

Remarks: you may or not use a polymorphic type for Church numbers. In this case, the representation can introduce a type abstraction (as in \( \lambda X.\lambda x : X.\lambda f : X \to X.f(f(f(x))) \)). Non-provability results are hard to obtain. A first attempt can be to consider the "proof-irrelevant" interpretation of the Calculus of Constructions. In this interpretation Prop is interpreted as a set with two values and proof dependencies in typed can be ignored.