Proof assistants

TD 2- Specifics of the Calculus of Constructions

1 Inductive Predicates

A- Give an inductive definition even : nat -> Prop for the predicate “to be even”.
B- Characterize with an inductive definition a relation exp : nat -> nat -> nat -> Prop with two constructors corresponding to the graph of the function \( n^p = q \) on natural numbers.

2 Recursive types

A- Propose in Coq an inductive definition with parameter corresponding to the ML type of polymorphic lists:

\[
\text{type} \ 'a\ \text{list} = \text{nil} \mid \text{cons} \ of \ 'a \ * \ 'a\ \text{list}
\]

B- Coq library defines the binary product, the unit type and the type of natural numbers:

\[
\text{Inductive} \ \text{prod} \ (A \ B : \text{Type}) : \text{Type} := \text{pair} : A \rightarrow B \rightarrow \text{prod} \ A \ B.
\]

\[
\text{Inductive} \ \text{unit} : \text{Type} := \text{tt} : \text{unit}.
\]

\[
\text{Inductive} \ \text{nat} : \text{Type} := O : \text{nat} \mid S : \text{nat} \rightarrow \text{nat}.
\]

Construct an expression prodn in CCI of type Type -> nat -> Type which builds the n-ary product of a given type \( A \): (i.e. prodn \( A \ n \) is \( A \times \ldots \times A \) (\( n \) times)). The definition will be by recursion on \( n \).

Give an expression length of type \( \forall A.\ \text{list} \ A \rightarrow \text{nat} \) which computes the length of a list.

Give an expression embed of type \( \forall A.\ \forall l : \text{list} \ A.\ \text{prodn} \ A \ (\text{length} \ l) \) which translates a list into a n-uple.

3 Termination of fixpoints

Are the following fixpoints well-founded in CCI? explain why?

\[
\text{Fixpoint} \ \text{leq} \ (n \ p : \text{nat}) \ \{\text{struct} \ n\} : \text{bool} :=
\]

\[
\text{match} \ n \ \text{with}
\]

\[
| O \Rightarrow \text{true} \mid S n' \Rightarrow \text{match} \ p \ \text{with} \ O \Rightarrow \text{false} \mid S p' \Rightarrow \text{leq} \ n' \ p' \end
\]

end.

\[
\text{Definition} \ \text{exp} \ (p: \text{nat}) :=
\]

\[
(\text{fix} \ f \ (n: \text{nat}) : \text{nat} :=
\]

\[
\text{match} \ \text{leq} \ p \ n \ \text{with} \ | \ \text{true} \Rightarrow S 0 \ | \ \text{false} \Rightarrow f \ (S \ n) + f \ (S \ n) \end
\]

end)

0.

\[
\text{Definition} \ \text{ackermann} := \text{fix} \ f \ (n: \text{nat}) : \text{nat} \rightarrow \text{nat} := \text{match} \ n \ \text{with}
\]

\[
| O \Rightarrow S \mid S n' \Rightarrow \text{fix} \ g \ (m: \text{nat}) : \text{nat} := \text{match} \ m \ \text{with}
\]

\[
| O \Rightarrow f \ n' \ (S \ 0) \mid S m' \Rightarrow f \ n' \ (g \ m') \end
\]

end.

4 Strong elimination

Let \( t_1 \) and \( t_2 \) be two arbitrary terms of type \( T_1 \) and \( T_2 \). Is the following function typable?

**Definition** \( g \ (b : \text{bool}) := \text{match} \ b \ \text{with} \ true \Rightarrow t_1 \ | \ false \Rightarrow t_2 \ \text{end} \).

If yes, give the corresponding return clause.

5 Restrictions on sorts in eliminations

A- We introduce the following definition of the true proposition:

**Inductive** \( \text{True} : \text{Prop} := \text{I} : \text{True} \).

Write a function from \( \text{unit} \) to \( \text{True} \) which is one-to-one.

B- We now introduce

**Inductive** \( \text{BOOL} : \text{Prop} := \text{TRUE} : \text{BOOL} | \text{FALSE} : \text{BOOL} \).

Can we show the equivalence between \( \text{bool} \) and \( \text{BOOL} \)? Show that such an equivalence gives a proof of the negation of the principle of proof-irrelevance \( \forall P : \text{Prop} \forall p q : P. p = q \).

6 The type \( W \) of well-founded trees (exam 2008)

The type \( W \) of well-founded trees is parameterised by a type \( A \) and a family of types \( B : A \rightarrow \text{Type} \). It has only one constructor and is defined by:

**Inductive** \( W \ (A : \text{Type}) \ (B : A \rightarrow \text{Type}) : \text{Type} := \)

\[ \text{node} \ : \ \forall (a : A), \ (B \ a \rightarrow W \ A \ B) \rightarrow W \ A \ B. \]

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 \( \text{Type} \).

2. In order to encode the type \( \text{nat} \) of natural numbers with \( 0 \) and \( S \), we need two types of nodes. We take \( A = \text{bool} \). The constructor \( 0 \) corresponds to \( a = \text{false} \), it does not expect any argument so we take \( B \text{false} = \text{empty} \). The constructor \( S \) corresponds to \( a = \text{true} \), it takes one argument, we define \( B \text{true} = \text{unit} \).

Using this encoding, give the terms corresponding to \( \text{nat} \), \( 0 \) et \( S \).

3. Propose an encoding using \( W \) for the type \( \text{tree} \) of binary trees parameterised by a type of values \( V \), which means that we have a constructor \( \text{leaf} \) of type \( (\text{tree} \ V) \) and a constructor \( \text{bin} \) of type \( \text{tree} \ V \rightarrow V \rightarrow \text{tree} \ V \rightarrow \text{tree} \ V \). 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 \( \forall x : \text{unit}, f_1 \ x = n \) 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 \( W \)? Propose an equality on the type \( W \) which solves this problem.