1 Basic inductive definitions

1.1 Booleans

We define

\begin{verbatim}
Inductive bool : Type := true : bool | false : bool.
\end{verbatim}

1- Define boolean negation `negb` and boolean conjunction `andb` in CCI.

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 \texttt{Eval compute in}). What is remarkable?

1.2 Logical connectives

Observe how the logical connectives and, or, ex and their induction schemes are defined in the standard library of Coq, using the command \texttt{Print ident}.

2 Recursive types

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

\begin{verbatim}
type 'a list = nil | cons of 'a × 'a list
\end{verbatim}

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

\begin{verbatim}
Inductive prod (A B : Type) : Type := pair : A → B → prod A B.
Inductive unit : Type := tt : unit .
\end{verbatim}

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 × ... × A` (\(n\) times)). The definition will be by recursion on \(n\).

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

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