1 Basic inductive definitions

1.1 Booleans

We define

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

1- Define boolean negation \textit{negb} and boolean conjunction \textit{andb} in CCI.

2- Detail the normalisation steps of expressions \( \lambda x : \text{bool} . \text{negb} \left( \text{andb} \text{false} x \right) \) et \( \lambda x : \text{bool} . \text{negb} \left( \text{andb} x \text{false} \right) \)

(in Coq, one can use the command \texttt{Eval compute in \ldots}). 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:

\[
\text{type 'a list = nil | cons of 'a \ast 'a list}
\]

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

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

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

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

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

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

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