Proof Assistants – TP. 3

Bruno Barras

Jan 5, 2017

1 Basic inductive definitions

1.1 Booleans

We define

```
Inductive bool : Type := true : bool | false : bool.
```

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

2- Detail the normalisation steps of expressions $\lambda x : \texttt{bool}.\, negb\ (andb\ false\ x)$ et $\lambda x : \texttt{bool}.\, negb\ (andb\ x\ false)$ (in Coq, one can use the command `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 `Print` *ident*.

2 Recursive types

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

```
type 'a list = nil | cons of 'a * 'a list
```

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

```
Inductive prod (A B : Type) : Type := pair : A → B → prod A B.
Inductive unit : Type := tt : unit.
Inductive nat : Type := O : nat | S : nat → nat.
```

Construct an expression `prodn` in CCI of type `Type` $\rightarrow$ `nat` $\rightarrow$ `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 $\forall A.$ `list` $A \rightarrow$ `nat` which computes the length of a list.

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