2.7.1 — Foundations of Proof Systems

Exam

2017-2018

Try to be as concise as possible. The aim is to show you understand what is going on. Syntactical details are for the computer, not for the paper.

1 Warming up...

Question 1 Give a proof in natural deduction of the following proposition :

$$(f \implies (g \implies h)) \implies ((f \implies g) \implies (f \implies h)).$$ ⋄

Question 2 Consider a closed term t in Martin-Löf type theory whose type is :

$$t : \forall\, n : \mathrm{nat}\,.\ \Sigma\, p : \mathrm{nat}\,.\ \mathrm{prime}\ \ p \wedge n < p \wedge (\forall\, k : \mathrm{nat}\,.\, n < k < p \to \neg\, \mathrm{prime}\, k).$$

where $\mathrm{prime}\ \ p$ is a predicate that holds iff p is a prime number.

Give the normal form of $\pi_1(t\ 8)$ where π_1 is the first projection for Σ-types. ⋄

Question 3 Give a closed term whose type is :

$$\forall (A\ B : \mathsf{Type})(P : A \to B \to \mathsf{Prop}).$$
$$(\Sigma\, y : B.\ \forall\, x : A.\, P\ x\ y) \to (\forall\, x : A.\ \Sigma\, y : B.\, P\ x\ y).$$ ⋄

2 Strong vs Weak Induction

Question 4 How would you prove in type theory the following scheme :

$$\forall (P : \mathrm{nat} \to \mathsf{Prop}).$$
$$(\forall (n : \mathrm{nat}).\, (\forall (p : \mathrm{nat}).\, p < n \to P\ p) \to P\ n) \to$$
$$\forall (n : \mathrm{nat}).\, P\ n$$

from the usual induction scheme :

$$\forall (P : \mathrm{nat} \to \mathsf{Prop}).$$
$$P\ 0 \to$$
$$(\forall (n : \mathrm{nat}).\, P\ n -> P\ (\mathrm{S}\ \ n)) \to$$
$$\forall (n : \mathrm{nat}).\, P\ n$$

You do not have to give a deduction tree or a proof-term. Describe the proof is a convincing way. The fact whether we are in arithmetic, Coq or HOL is not very relevant here. ⋄

3 Limited Principle of Omniscience

We define the following three propositions.

$$\mathrm{EM} \triangleq \forall(P : \mathsf{Prop}).\, P \lor \neg P$$

$$\mathrm{LPO} \triangleq \forall(P : \mathrm{nat} \to \mathrm{bool}).\, (\forall(n : \mathrm{nat}).\, P\,n = \bot) \lor (\Sigma(n : \mathrm{nat}).\, P\,n = \top)$$

$$\begin{aligned}\mathrm{LLPO} \triangleq \forall(P : \mathrm{nat} \to \mathrm{bool}).\\ &(\forall(n\,p : \mathrm{nat}).\, P\,n = \top \land P\,p = \top \to n = p) \to\\ &(\forall(i : \mathrm{nat}).\, P(2i) = \bot) \lor (\forall(i : \mathrm{nat}),\, P(2i+1) = \bot)\end{aligned}$$

Question 5 Prove constructively that EM → LPO and that LPO → LLPO. (same remark regarding the level of detail as for the previous question.) ◇

We now give a variant of LPO where the predicates in consideration are not necessarily decidable :

$$\mathrm{LPPO} \triangleq \forall(P : \mathrm{nat} \to \mathsf{Prop}).\, (\forall(n : \mathrm{nat}).\, \neg P\,n) \lor (\Sigma(n : \mathrm{nat}).\, P\,n)$$

Question 6 Prove that LPPO → EM. ◇

4 Being even

We remind the usual definition of addition in Coq :

```
Fixpoint add n m :=
 match n with
  | 0 => m
  | S p => S (add p m)
end.
```

The following is a possible definition of the property of being even in Coq :

```
Inductive even : nat -> Prop :=
| E0 : even 0
| ESS : forall n, even n -> even (S (S n)).
```

Question 7 What is the elimination scheme associated to this definition ? ◇

A friend comes up with the following alternative definition :

```
Definition evs (n : nat) : Prop :=
  exists x, n = x + x.
```

Question 8 We want to show that `forall n, even n -> evs n`. What would the main steps be ? ◇

Question 9 Conversely, how would you prove `forall n, evs n -> even n.` ◇

5 Recursive and inductive predicates

We consider the usual definition of natural numbers in Coq (with constructors `O` and `S`) and the following (usual) definition of lists :

```
Inductive list : Type :=
  nil : list
| cons : nat -> list -> list.
```

Question 10 Here are four predicates defined by case analysis and recursion. For each of them, give an equivalent inductive predicate of the same type. You do not have to give the proof that your formulation is equivalent to the given predicate. However, *try to give an inductive predicate that is as concise and elegant as possible*. Every time, give also the type of the generated elimination principle.

```
Fixpoint N1 (n : nat) : Prop :=
  match n with
  | O => True
  | S O => False
  | S (S O) => False
  | S (S (S m)) => N1 m
  end.

Fixpoint L1 (l : list) : Prop :=
  match l with
  | nil => True
  | cons n l' => (n1 n)/\(L1 l')
  end.

Fixpoint L2 (n:nat)(l:list) : Prop :=
  match l with
  | nil => False
  | cons m l' => n=m \/ L2 n l'
  end.


Fixpoint addChain (l : list) : Prop :=
  match l with
  | nil => False
  | cons (S O) nil => True
  | cons _ nil => False
  | cons m l' => exists x, exists y, (L2 x l')/\(L2 y l')/\m=x+y/\addChain l'
  end.
```

Question 11 (optional) The last predicate characterizes so called *addition chains*. An addition chain ending with number n gives a way to compute a^n in a time proportional to the length of the list. Can you see why? ⋄

(Finding the shortest addition chain ending with n is an open problem. There is no known reasonably efficient algorithm.)