CSE-433 Assignment 8 - Coq Programming (VII)
(100 points)

gla@postech

Due at 11am, Dec 4, Tuesday

In this assignment, you will practice all the commands and tactics in Coq that you have learned so far. Please do not discuss the assignment with your classmates. You are, however, encouraged to post on the discussion board any questions you might have about Coq.

Please do not use the `auto` tactic or any similar tactics.

1 Strings of matched parentheses (50 points)

The goal of this part is to translate the proofs of Theorem 1.9 and its converse (in Exercise 1.16) given in the Course Notes. That is, we use the following inference rules to prove the two theorems shown below:

$$\frac{}{\epsilon\ \mathsf{mparen}}\ \mathit{Meps} \qquad \frac{s\ \mathsf{mparen}}{(s)\ \mathsf{mparen}}\ \mathit{Mpar} \qquad \frac{s_1\ \mathsf{mparen} \quad s_2\ \mathsf{mparen}}{s_1\ s_2\ \mathsf{mparen}}\ \mathit{Mseq}$$

$$\frac{}{\epsilon\ \mathsf{lparen}}\ \mathit{Leps} \qquad \frac{s_1\ \mathsf{lparen} \quad s_2\ \mathsf{lparen}}{(s_1)\ s_2\ \mathsf{lparen}}\ \mathit{Lseq}$$

Theorem 1.1. *If* s mparen*, then* s lparen.

Theorem 1.2. *If* s lparen*, then* s mparen.

We provide a definition for strings of parentheses (`S`) and a function for concatenating two strings of parentheses (`concat`). Your task is to define two inductive judgments `mparen` and `lparen` according to the inference rules shown above, and to give proofs of theorems `mparen2lparen` and `lparen2mparen`.

```
Inductive E : Set :=
| LP : E
| RP : E.

Inductive S : Set :=
| eps : S
| cons : E -> S -> S.

Fixpoint concat (s1 s2:S) {struct s1} : S :=
match s1 with
| eps => s2
| cons e s2' => cons e (concat s2' s2) end.

Inductive mparen : S -> Prop := ...

Inductive lparen : S -> Prop := ...

Theorem mparen2lparen : forall s:S, mparen s -> lparen s.

Theorem lparen2mparen : forall s:S, lparen s -> mparen s.
```

You may introduce additional lemmas to simplify the proof. You may also need to prove some properties of `concat`, *e.g.*, `concat s eps = s`. Feel free to introduce any auxiliary definitions that are necessary to complete the proofs. All that I care about is your definitions of `mparen` and `lparen` and your proofs of `mparen2lparen` and `lparen2mparen`.

2 Complete induction (50 points)

In Exercise 1.21 in the Course Notes, we have learned the principle of complete induction, which appears to be more powerful than mathematical induction, but turns out to be a derived notion. In this part, you will give a proof of the principle of complete induction in Coq. The goal is to give a proof of the theorem `nat_complete_ind` shown below:

```
Inductive nat : Set :=
| O : nat
| S : nat -> nat.

Inductive lt : nat -> nat -> Prop :=
| lt_O : forall n:nat, lt O (S n)
| lt_S : forall (m:nat) (n:nat), lt m n -> lt (S m) (S n).

Variable P : nat -> Prop.

Theorem nat_complete_ind :
  P O -> (forall n:nat, (forall z:nat, lt z n -> P z) -> P n) -> forall x:nat, P x.
```

The theorem can be written in our notation as follows:

$$P(\mathbf{0}) \supset (\forall n \in \mathsf{nat}.(\forall z \in \mathsf{nat}.z < n \supset P(z)) \supset P(n)) \supset \forall x \in \mathsf{nat}.P(x) \ \mathit{true}$$

Here are a few hints that you might find useful.

- Remember that complete induction is a principle derived from mathematical induction. This implies that your proof should contain an application of `nat_ind` somewhere.
- Then the whole problem boils down to finding an appropriate predicate, say `A n` where `n` is a natural number, for the application of `nat_ind`. Then this application of `nat_ind` will prove `forall n:nat, A n`. This is the key part of your proof.
- `A n` should *not* be `P n`. Instead you have to generalize the goal statement so that `forall n:nat, A n` would *imply* `forall n:nat, P n`. Letting `A n = P n` will fail!
- Before starting to write a proof in Coq, try to find a mathematical proof. Without a solid understanding of how the proof works, it might be very difficult to complete the proof in Coq in an interactive manner. That is, Coq helps you a lot especially when you know how to complete the proof yourself.
- You can simplify the presentation by explicitly defining the predicate `A`, as in:

  ```
  Let A : nat -> Prop := fun k:nat => ...
  ```
- This is a line copied directly from the sample solution:

  ```
  apply (nat_ind A AO (Aind H)).
  ```
- You will have to prove some properties of `lt`.

Submission instruction

Download the stub file `coq7.v` from the course webpage and complete all the proofs in it. Then copy it to your hand-in directory. For example, if your Hemos ID is `foo`, copy it to:

```
/afs/postech.ac.kr/class/cse/cs433/handin/hw8/foo/
```