CIS 500 — Software Foundations
Midterm I

Review questions

October 10, 2007

The following problems should give you a general idea of the style and coverage of the exam questions. Note that this is not a sample exam: in particular, we have not tried to tune the number of questions here so that they can be finished by most people in 80 minutes (which we will do for the test).
1. Consider the following Fixpoint definition in Coq.

\[
\text{Fixpoint fold (X:Set) (Y:Set) (f:X->Y->Y) (l:list X) (b:Y) \{struct l\} : Y :=}
\]
\[
\text{match l with}
\]
\[
\text{| nil => b}
\]
\[
\text{| h :: t => f h (fold _ _ f t b)}
\]
\[
\text{end.}
\]

What is the type of fold? (I.e., what does \text{Check fold print?})

2. What does \text{Check fold _ _ plus. print?}

3. What does \text{Eval simpl in (fold _ _ plus [one,two,three,four] one). print?}

4. Recall that the map function takes a function f of type X->Y and a list l with elements of X and returns a list with elements of type Y containing the result of applying f to each element of l.

Which of the following Fixpoint definitions correctly implements the map function? Circle the correct answer.

(a) Fixpoint m1 (X:Set) (Y:Set) (f:X->Y) (l:list X) \{struct l\} : (list Y) :=
\[
\text{match l with}
\]
\[
\text{| nil => nil Y | h :: t => h :: (m1 X Y f t) end.}
\]

(b) Fixpoint m2 (X:Set) (Y:Set) (f:X->Y) (l:list X) \{struct l\} : (list Y) :=
\[
\text{match l with}
\]
\[
\text{| nil => f (nil Y) | h :: t => (f h) :: (m2 X Y f t) end.}
\]

(c) Fixpoint m3 (X:Set) (Y:Set) (f:X->Y) (l:list X) \{struct l\} : (list Y) :=
\[
\text{match l with}
\]
\[
\text{| nil => nil Y | h :: t => (f h) :: (m3 X Y f t) end.}
\]

(d) Fixpoint m4 (X:Set) (Y:Set) (f:X->Y) (l:list X) \{struct l\} : (list Y) :=
\[
\text{match l with}
\]
\[
\text{| nil => nil Y | h :: t => f h :: f (m4 X Y f t) end.}
\]

(e) Fixpoint m5 (X:Set) (Y:Set) (f:X->Y) (l:list X) \{struct l\} : (list Y) :=
\[
\text{match l with}
\]
\[
\text{| nil => nil Y | h :: t => f h :: f t end.}
\]

(f) None of the above.
5. The \texttt{every} function takes a predicate \( p \) (a one-argument function returning a \texttt{yesno}) and a list \( l \); it returns \texttt{yes} if \( p \) returns \texttt{yes} on every element of \( l \) and \texttt{no} otherwise.

\begin{verbatim}
Lemma check_filter1:
  every nat even [two,four,zero] = yes.
Proof. simpl. reflexivity. Qed.

Lemma check_filter2:
  every nat even [two,one,four,zero] = no.
Proof. simpl. reflexivity. Qed.
\end{verbatim}

What is the type of \texttt{every}?

6. Complete the following definition of \texttt{every} as a recursive function:

\begin{verbatim}
Fixpoint every (X:Set) (p:X->yesno) (l:list X) {struct l} : yesno :=
  match l with
  | nil => ____________________
  | h :: t => both_yes ____________________________ ________________________________
  end.
\end{verbatim}

7. Complete the following definition of \texttt{every} by supplying appropriate arguments to \texttt{fold}:

\begin{verbatim}
Definition every' (X:Set) (p:X->yesno) (l:list X) : yesno :=
  fold _ _
  (fun x acc => both_yes ___________ ___________) __________ __________.
\end{verbatim}
Coq Basics

8. Briefly explain the difference between the tactics apply and rewrite. Are there situations where either one can be applied?

9. Briefly explain the difference between the tactics destruct and induction.

10. The following proof attempt is not going to succeed. Briefly explain why and how it can be fixed.

```coq
Lemma double_injective : forall m n, 
  double m = double n 
  -> m = n.
Proof. 
  intros m n. induction m. 
  Case "O". simpl. intros eq. destruct n. 
    Case "O". reflexivity. 
    Case "S". inversion eq. 
  Case "S". intros eq. destruct n. 
    Case "O". inversion eq. 
    Case "S". 
      assert (m = n) as H. 
      Case "Proof of assertion". 
```

3
11. Suppose we give Coq the following declarations:

\[
\text{Inductive mumble : Set :=} \\
\quad | \ a : \text{mumble} \\
\quad | \ b : \text{mumble} \to \text{nat} \to \text{mumble} \\
\quad | \ c : \text{mumble}. \\
\]

\[
\text{Inductive grumble (X:Set) : Set :=} \\
\quad | \ d : \text{mumble} \to \text{grumble X} \\
\quad | \ e : X \to \text{grumble X}. \\
\]

Which of the following are well-typed members of the set \text{grumble}?

(a) \(d \ (b \ a \ five)\)

(b) \(d \ \text{mumble} \ (b \ a \ five)\)

(c) \(d \ \text{yesno} \ (b \ a \ five)\)

(d) \(e \ \text{yesno} \ \text{yes}\)

(e) \(e \ \text{mumble} \ (b \ c \ \text{zero})\)

(f) \(e \ \text{yesno} \ (b \ c \ \text{zero})\)

(g) \(c\)

12. Consider the following inductive definition:

\[
\text{Inductive baz : Set :=} \\
\quad | \ x : \text{baz} \to \text{baz} \\
\quad | \ y : \text{baz} \to \text{yesno} \to \text{baz}. \\
\]

How many elements does the set \text{baz} have?
13. Consider the following inductive definition:

\[
\text{Inductive tree : Set :=}
| \text{leaf : tree}
| \text{node : tree \to tree \to tree.}
\]

Describe, in English, the elements of the set \text{tree}.

14. What induction principle will Coq generate for \text{tree}?
15. Here is an induction principle for an inductively defined set \( s \).

\[
\text{myset}_\text{ind} : \\
\quad \text{forall } P : \text{myset }\to \text{Prop}, \\
\qquad (\text{forall } y : \text{yesno}, P (\text{con1 } y)) \\
\qquad \quad \to (\text{forall } n : \text{nat} (m : \text{myset}), P m \to P (\text{con2 } n m)) \\
\qquad \quad \to \text{forall } m : \text{myset}, P m
\]

What is the definition of \text{myset}?

16. Consider the following inductive definition:

\[
\text{Inductive stree} : \text{nat }\to \text{Set} := \\
\text{\hspace{1em} | leaf : stree \text{one} \\
\text{\hspace{1em} | node : forall m n, stree m }\to\text{ stree n }\to \text{ stree (plus m n).}
\]

Describe, in English, the elements of the set \text{stree seven}. 

17. Consider the following inductively defined family of propositions:

\[
\text{Inductive } p : \text{tree} \rightarrow \text{nat} \rightarrow \text{Prop} := \\
\mid c1 : p \text{ leaf one} \\
\mid c2 : \forall t1 t2 n1 n2, \\
\hspace{1em} p t1 n1 \rightarrow p t2 n2 \rightarrow p (\text{node } t1 t2) (\text{plus } n1 n2).
\]

Describe, in English, the conditions under which the proposition \( p t n \) is provable.

18. Consider the following inductively defined family of propositions:

\[
\text{Inductive } \text{bar} : \text{nat} \rightarrow \text{Prop} := \\
\mid d : \text{bar six} \\
\mid e : \forall n, \text{bar } (\text{times } n n) \\
\mid f : \text{bar three } \rightarrow \text{bar five}.
\]

For which \( n \) is the proposition \( \text{bar } n \) provable?
19. The concept of composition of relations can be defined as follows:

Suppose \( Q \) and \( R \) are both relations on a set \( X \). The composition of \( Q \) and \( R \) is the relation \( C \) such that, for all \( x \) and \( z \) in \( X \),

\[
C x z \iff \exists y. Q x y \land R y z.
\]

Write an Inductive definition in Coq that expresses this concept.

20. Give the definition of logical conjunction (and) as an inductive proposition in Coq.
21. Recall the definition of the set $tm$, the predicate $value$, and the relation $eval$ from the lecture notes:

\[
\text{Inductive } tm : \text{Set} := \\
\mid tm\_const : \text{nat} \rightarrow tm \\
\mid tm\_plus : tm \rightarrow tm \rightarrow tm.
\]

\[
\text{Inductive } value : tm \rightarrow \text{Prop} := \\
\mid v\_const : \forall n, \text{value } (tm\_const n).
\]

\[
\text{Inductive } eval : tm \rightarrow tm \rightarrow \text{Prop} := \\
\mid E\_PlusConstConst : \forall n1 n2, \\
\quad \text{eval } (tm\_plus (tm\_const n1) (tm\_const n2)) \\
\quad \quad (tm\_const (\text{plus } n1 n2)) \\
\mid E\_Plus1 : \forall t1 t1' t2, \\
\quad \text{eval } t1 t1' \\
\quad \quad \rightarrow \text{eval } (tm\_plus t1 t2) \\
\quad \quad \quad (tm\_plus t1' t2) \\
\mid E\_Plus2 : \forall v1 t2 t2', \\
\quad \text{value } v1 \\
\quad \quad \rightarrow \text{eval } t2 t2' \\
\quad \quad \quad \rightarrow \text{eval } (tm\_plus v1 t2) \\
\quad \quad \quad \quad (tm\_plus v1 t2').
\]

Which of the following propositions are provable?

(a) \[\text{eval } (tm\_plus (tm\_const one) (tm\_const two)) \\
\quad (tm\_const (plus one two)).\]

(b) \[\text{eval } (tm\_plus \\
\quad (tm\_plus (tm\_const one) (tm\_const two)) \\
\quad (tm\_const three)) \\
\quad (tm\_const (plus (plus one two) three)).\]

(c) \[\text{eval } (tm\_plus \\
\quad (tm\_plus (tm\_const one) (tm\_const two)) \\
\quad (tm\_plus (tm\_const three) (tm\_const four))) \\
\quad (tm\_plus \\
\quad (tm\_plus (tm\_const one) (tm\_const two)) \\
\quad (tm\_const (plus three four))).\]
Recall the determinism and progress theorems for the eval relation.

Theorem eval_deterministic :
  partial_function _ eval.

Theorem eval_progress : forall t,
  value t / (exists t', eval t t').

22. (a) Suppose we add a new constructor to the evaluation relation as follows:

| E_Funny1 :
  eval (tm_const zero)
  (tm_const zero)

i. Does eval_deterministic continue to hold?

ii. Does eval_progress continue to hold?

(b) Suppose we remove the constructor v_const from the definition of value.

i. Does eval_deterministic continue to hold?

ii. Does eval_progress continue to hold?

(c) Is there any way we can cause eval_progress to fail by adding new constructors to the definition of eval?