CIS 500 — Software Foundations
Midterm I

February 18, 2009

<table>
<thead>
<tr>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

Name: __________________________________________

Email: _________________________________________
Instructions

• This is a closed-book exam: you may not use any books or notes.

• You have 80 minutes to answer all of the questions.

• The exam is worth 80 points. However, questions vary significantly in difficulty, and the point value of a given question is not always exactly proportional to its difficulty. Do not spend too much time on any one question.

• Partial credit will be given. All correct answers are short. The back side of each page may be used as a scratch pad.

• Good luck!
1. (5 points) Consider the following Coq function:

```coq
Fixpoint concatMap (X Y : Set) (f : X → list Y) (l : list X) {struct l} : list Y :=
  match l with
  | nil => nil
  | h :: t => (f h) ++ (concatMap _ _ f t)
end.
```

(a) What is the type of `concatMap`? (I.e., what does `Check concatMap` print?)

(b) What does

```
Eval simpl in (concatMap _ _ (fun x => x) [[1,2],[3,4]]).
```

print?

(c) What does

```
Eval simpl in (concatMap _ _ (fun x => [x+1,x+2]) ([1,2])).
```

print?
2. (5 points)

(a) Fill in the definition of the Coq function \texttt{elem} below.

Given a type \(X\), an equality-testing function \(\text{eq}\) for \(X\), an element \(e\) of type \(X\), and a list \(l\) of type \(\text{list } X\), the expression \(\text{elem } X \text{ eq } e \text{ l}\) returns \texttt{true} if and only an element \(\text{eq\text{-}equal}\) to \(e\) appears in the list. For example, \(\text{elem } \text{nat } \text{beq\_nat } 2 \ [1,2,3]\) yields \texttt{true} (because \(\text{beq\_nat } 2 \ 2 = \texttt{true}\)) while \(\text{elem } \text{nat } \text{beq\_nat } 5 \ [1,2,3]\) yields \texttt{false}.

Fixpoint \texttt{elem} \(\text{X : Set)} \ (\text{eq} : \text{X} \rightarrow \text{X} \rightarrow \text{bool}) \ (e : \text{X}) \ (l : \text{list } X)\)
\(
\{\text{struct } l\} : \text{bool :=}
\)

(b) Why do we need to pass an equality-testing function \(\text{eq}\) as an argument to \texttt{elem} instead of just using \(=\) to test for equality?

3. (6 points) Fill in the definition of the Coq function \texttt{nub} below.

Given a type \(X\), an equality function \(\text{eq}\) for \(X\), and a list \(l\) of type \(\text{list } X\), the expression \(\text{nub } X \text{ eq } l\) yields a list that retains only the last copy of each element in the input list. For example, \(\text{nub } \text{nat } \text{beq\_nat } [1,2,1,3,2,2,4]\) yields \(1,3,2,4\).

Fixpoint \texttt{nub} \(\text{X : Set)} \ (\text{eq} : \text{X} \rightarrow \text{X} \rightarrow \text{bool}) \ (l : \text{list } X)\)
\(
\{\text{struct } l\} : \text{list } X :=
\)
4. (5 points)
   (a) Briefly explain the use and behavior of the apply tactic.

   (b) Briefly explain the use and behavior of the apply ... in ... tactic.
5. (6 points) Recall the Coq function \texttt{repeat}:

\begin{verbatim}
Fixpoint repeat (X : Set) (n : X) (count : nat) {struct count} : list X :=
match count with
| 0 => nil
| S count' => cons n (repeat _ n count')
end.
\end{verbatim}

Consider the following partial proof:

\begin{verbatim}
Lemma repeat_injective : forall (X : Set) (x : X) (n m : nat),
repeat _ x n = repeat _ x m →
n = m.
Proof.
intros X x n m eq. induction n as \\
|n'|.
Case "n = 0". destruct m as [|m'].
SCase "m = 0". reflexivity.
SCase "m = S m'". inversion eq.
Case "n = S n'". destruct m as [|m'].
SCase "m = 0". inversion eq.
SCase "m = S m'".
assert (n' = m') as H.
SCase "Proof of assertion".
\end{verbatim}

Here is what the "goals" display looks like after Coq has processed this much of the proof:

\begin{verbatim}
2 subgoals
SSCase := "Proof of assertion" : String.string
SCase := "m = S m'" : String.string
Case := "n = S n'" : String.string
X : Set
x : X
n' : nat
m' : nat
eq : repeat X x (S n') = repeat X x (S m')
IHn' : repeat X x n' = repeat X x (S m') → n' = S m'

subgoal 2 is:
S n' = S m'
\end{verbatim}

This proof attempt is not going to succeed. Briefly explain why and say how it can be fixed. (Do not write the repaired proof in detail—just say briefly what needs to be changed to make it work.)
6. (5 points) Suppose we make the following inductive definition:

```coq
Inductive foo (X : Set) (Y : Set) : Set :=
  | foo1 : X → foo X Y
  | foo2 : Y → foo X Y
  | foo3 : foo X Y → foo X Y.
```

Fill in the blanks to complete the induction principle that will be generated by Coq.

```coq
foo_ind :
  : forall (X Y : Set) (P : foo X Y → Prop),
  (forall x : X, __________________________________)
  →
  (forall y : Y, __________________________________)
  →
  (___________________________________________) →
  ____________________________________________
```

7. (6 points)

Consider the following induction principle:

```coq
bar_ind :
  : forall P : bar → Prop,
  (forall n : nat, P (bar1 n)) →
  (forall b : bar, P b → P (bar2 b)) →
  (forall (b : bool) (b0 : bar), P b0 → P (bar3 b b0)) →
  forall b : bar, P b
```

Write out the corresponding inductive set definition.

```coq
Inductive bar : Set :=
  | bar1 : _______________________________
  | bar2 : _______________________________
  | bar3 : _______________________________.
```
8. (6 points) Suppose we give Coq the following definition:

\[
\text{Inductive } R : \text{nat} \rightarrow \text{list nat} \rightarrow \text{Prop} := \\
| c1 : R 0 [\] \\
| c2 : \forall n \text{l}, R n \text{l} \rightarrow R (\text{S} n) (\text{n} :: \text{l}) \\
| c3 : \forall n \text{l}, R (\text{S} n) \text{l} \rightarrow R n \text{l}.
\]

Which of the following propositions are provable? (Write \textit{yes} or \textit{no} next to each one.)

(a) \(R \ 2 \ [1,0]\)

(b) \(R \ 1 \ [1,2,1,0]\)

(c) \(R \ 6 \ [3,2,1,0]\)

9. (6 points) The following inductively defined proposition...

\[
\text{Inductive } \text{appears\_in (X:Set)} \ (a:X) : \text{list X} \rightarrow \text{Prop} := \\
| \text{ai\_here} : \forall \text{l}, \text{appears\_in X a (a::l)} \\
| \text{ai\_later} : \forall \text{b l}, \text{appears\_in X a l} \rightarrow \text{appears\_in X a (b::l)}.
\]

...gives us a precise way of saying that a value \(a\) appears at least once as a member of a list \(l\).

Use \text{appears\_in} to complete the following definition of the proposition \text{no\_repeats X l}, which should be provable exactly when \(l\) is a list (with elements of type \(X\)) where every member is different from every other. For example, \text{no\_repeats nat [1,2,3,4]} and \text{no\_repeats bool []} should be provable, while \text{no\_repeats nat [1,2,1]} and \text{no\_repeats bool [true,true]} should not be.

\[
\text{Inductive } \text{no\_repeats (X:Set)} : \text{list X} \rightarrow \text{Prop} :=
\]
10. (2 points) Complete the definition of \texttt{and}, as it is defined in \texttt{Logic.v}:

\begin{verbatim}
Inductive and (A B : Prop) : Prop :=
\end{verbatim}

11. (2 points) Complete the definition of \texttt{or}, as it is defined in \texttt{Logic.v}:

\begin{verbatim}
Inductive or (A B : Prop) : Prop :=
\end{verbatim}

12. (6 points) Write an informal proof (in English) of the proposition $\forall P : Prop, \neg(P \land \neg P)$. 
13. (4 points) Recall the nat-indexed proposition \( ev \) from Logic.v:

\[
\text{Inductive } ev : \text{nat} \rightarrow \text{Prop} := \\
| ev_0 : ev 0 \\
| ev_SS : \forall n : \text{nat}, ev n \rightarrow ev (S (S n)).
\]

Complete the definition of the following proof object:

\[
\text{Definition ev_plus2 : } \forall n, ev n \rightarrow ev (\text{plus} 2 n) :=
\]

14. (6 points) Recall the definition of \( \text{ex} \) (existential quantification) from Logic.v:

\[
\text{Inductive } \text{ex} \ (X : \text{Set}) \ (P : X \rightarrow \text{Prop}) : \text{Prop} := \\
\quad \text{ex_intro : } \forall \text{witness} : X, P \text{ witness} \rightarrow \text{ex } X \ P.
\]

(a) In English, what does the proposition

\[
\text{ex nat (fun n => ev (S n))}
\]

mean?

(b) Complete the definition of the following proof object:

\[
\text{Definition p : ex nat (fun n => ev (S n)) :=}
\]
15. (10 points) Recall the definition of the index function:

\[
\text{Fixpoint index (X : Set) (n : nat) (l : list X) \{struct l\} : option X :=}
\]
\[
\text{match l with}
\]
\[
\text{| [] => None}
\]
\[
\text{| a :: l' => if beq_nat n 0 \then\ Some a \else index _ (pred n) l' \end.}
\]

Write an informal proof of the following theorem:

\[\forall X \, n \, l, \text{length } l = n \to \text{index } X (S \, n) \, l = \text{None}.\]