CIS 500: Software Foundations

Midterm I

October 3, 2017

Name (printed): 

Username (PennKey login id): 

My signature below certifies that I have complied with the University of Pennsylvania’s Code of Academic Integrity in completing this examination.

Signature: 
Date: 

Directions: This exam booklet contains both the standard and advanced track questions. Questions with no annotation are for both tracks. Other questions are marked “Standard Only” or “Advanced Only.” Do not waste time answering questions intended for the other track.

Put an X in the box of the track you are following. (If you are following one track but want to move to the other now, please mark the box for the track you want to be on and write a note on this page telling us that we should switch you to this track.)

☐ Standard ☐ Advanced
1. (7 points) Put an X in the True or False box for each statement.

(a) If the goal in the current proof state is \(3 \leq 3\), then using the tactic \texttt{reflexivity} will solve the goal.

□ True □ False

(b) For any \(f\), if \(H : f \ x = f \ y\) is an assumption in the current context, then \texttt{inversion H} will discover that \(x = y\).

□ True □ False

(c) If \(X\) is an inductively defined type or proposition with no constructors and \(\text{foo : X}\), then \texttt{destruct foo} will finish any proof or subgoal.

□ True □ False

(d) There are no expressions of type \texttt{False}.

□ True □ False (We threw out this question during grading: since it didn’t specify “in the empty context,” it’s hard to know whether the right answer is True or False!)

(e) The function \texttt{fun (X : Type) (x : X) (f : X -> X) => f x} terminates on all inputs.

□ True □ False

(f) A boolean function \(f : \texttt{nat -> bool}\) reflects a property \(P\) of numbers (\(P : \texttt{nat -> Prop}\)) exactly when \(\forall (n:\texttt{nat}), (f \ n = \texttt{true}) \iff P \ n\).

□ True □ False

(g) If \(E\) has type \texttt{beq_nat m n = true}, then \(E\) also has type \(m = n\).

□ True □ False
2. (14 points) Write the type of each of the following Coq expressions (write “ill-typed” if an expression does not have a type).

(a) \(4 \leq 3\)

(b) \(\forall (A : \text{Type}) (m \, n : A), \, m = n \lor m \neq n\)

(c) \(\text{fun} \, (x : \text{nat}) \Rightarrow \text{False}\)

(d) \(\forall (m : \text{nat}), \, m \times m\)

(e) \(\text{beq_nat} \, 3\)

(f) \(\text{fun} \, (P \, Q : \text{Prop}) \Rightarrow P \Rightarrow Q\)

(g) \(\text{fun} \, (m : \text{nat}) \, (E : 0 \leq m) \Rightarrow \text{le_S} \, 0 \, m \, E\)
3. [Standard Only] (16 points) For each of the types below, write a Coq expression that has that type or write “Empty” if there are no such expressions.

(a) \(\text{nat} \to \text{nat} \to \text{bool}\)

(b) \(\forall (X \ Y : \text{Type}), \text{list} \ X \to \text{list} \ Y\)

(c) \(\forall (X \ Y : \text{Type}) \ (f : X \to Y), Y\)

(d) \(\text{Prop} \to \text{bool}\)

(e) \(\text{In} \ 2 \ [1;1;1]\)

(f) \(\text{ev} \ 1\)

(g) \(\forall \ n : \text{nat}, \text{ev} \ n \to \text{ev} \ (S \ (S \ n))\)

(h) \((\text{nat} \to \text{nat}) \to \text{nat}\)
4. (18 points) In this problem, your job is to choose, for each of the following propositions, the best way of beginning a Coq proof to show that it is true.

(N.b. This problem was quite hard, as phrased. We dropped parts (d) and (e) during grading.)

(a) Suppose we are asked to prove the following:

\textbf{Theorem foo :} \forall n m, \ \n + 2 = m + 2 \rightarrow \ n = m.

Which of the following would be the best way to begin the proof?

- \hspace{1em} \square \ \text{intros n. destruct n.}
- \hspace{1em} \square \ \text{intros n. induction n.}
- \hspace{1em} \square \ \text{intros n m. induction n.}
- \hspace{1em} \square \ \text{intros n m. induction m.}
- \hspace{1em} \square \ \text{Not provable.}

(b) Suppose we are asked to prove the following:

\textbf{Theorem baz :} \forall A B (f g : A \rightarrow B), f = g \rightarrow (\forall x, f x = g x).

Which of the following would be the best way to begin the proof?

- \hspace{1em} \square \ \text{intros A B f g H. apply H.}
- \hspace{1em} \square \ \text{intros A B f g H. reflexivity.}
- \hspace{1em} \square \ \text{intros A B f g H. rewrite H.}
- \hspace{1em} \square \ \text{intros A B f g H x. induction x.}
- \hspace{1em} \square \ \text{Not provable.}

(c) Suppose we are asked to prove the following:

\textbf{Theorem bar :} \forall (P Q : \text{Prop}), P \rightarrow Q.

Which of the following would be the best way to begin the proof?

- \hspace{1em} \square \ \text{intros P Q H. induction P.}
- \hspace{1em} \square \ \text{intros P Q H. induction Q.}
- \hspace{1em} \square \ \text{intros P Q H. apply H.}
- \hspace{1em} \square \ \text{intros P Q H. apply ex_falso_quodlibet.}
- \hspace{1em} \square \ \text{Not provable.}
(d) Suppose we are asked to prove the following:

\[ \text{Theorem quux : } \forall (n : \text{nat}) \ s, \ \text{In } n \ s \rightarrow \text{In } n \ (s \ ++ \ s). \]

Which of the following would be the best way to begin the proof?

- \( \square \) intros n s H. left.
- \( \square \) intros n s H. right.
- \( \square \) intros n s H. apply H.
- \( \square \) intros n s H. destruct H.
- \( \square \) intros n s H. induction H.
- \( \square \) assert (forall (n:nat) s s', In n s -> In n (s ++ s')).
- \( \square \) Not provable.

(e) Suppose we are asked to prove the following (the inductive definition of \(1\text{e}\) can be found on page 1 of the appendix):

\[ \text{Theorem example : } \forall m n o, \ m \leq n \rightarrow n \leq o \rightarrow m \leq o. \]

Which of the following would be the best way to begin the proof?

- \( \square \) intros m n o H1. induction H1.
- \( \square \) intros m n o H1 H2. induction H1.
- \( \square \) intros m n o H1 H2. induction H2.
- \( \square \) Not provable.

(f) Suppose we are asked to prove the following (the definition of the regular-expression matching relation \( ^\approx \) can be found on page 2 of the appendix):

\[ \text{Lemma random : } \forall T (s : \text{list } T) (\text{re1 re2} : @\text{reg\_exp } T), \]
\[ \quad s \ ^\approx \text{re1} \ \lor \ s \ ^\approx \text{re2} \rightarrow \]
\[ \quad s \ ^\approx \ \text{Union re1 re2}. \]

Which of the following would be the best way to begin the proof?

- \( \square \) intros T s re1 re2 H. induction re1.
- \( \square \) intros T s re1 re2 H. induction re2.
- \( \square \) intros T s re1 re2 H. destruct H.
- \( \square \) intros T s re1 re2 H. induction s.
- \( \square \) Not provable.
5. This problem asks you to translate mathematical ideas from English into Coq.

(a) (5 points) The concept of “prefix of a list” can be defined as an inductive relation as follows:

\[
\text{Inductive prefix \{X : Type\} : list X \rightarrow list X \rightarrow Prop :=} \\
\mid \text{prefix\_nil : forall l, prefix \[\]\ l} \\
\mid \text{prefix\_cons : forall t l h, prefix t l \rightarrow prefix (h::t) (h::l).}
\]

For example, these are all the prefixes of \([1;2;3]\) (i.e., all the lists \(ls\) such that \(\text{prefix\ ls\ [1;2;3]}\) is provable):

\[
\begin{align*}
\[] \\
[1] \\
[1;2] \\
[1;2;3]
\end{align*}
\]

Conversely, a \textit{suffix} of a list is a substring that occurs at the end of the larger string. For example, here are all the suffixes of \([1;2;3]\):

\[
\begin{align*}
\[] \\
[3] \\
[2;3] \\
[1;2;3]
\end{align*}
\]

Complete the following inductive definition of the suffix relation, where \texttt{suffix s l} indicates that \(s\) is a suffix of \(l\).

\[
\text{Inductive suffix \{A : Type\} : list A \rightarrow list A \rightarrow Prop :=}
\]
(b) (8 points) A binary tree with labels of type \( A \) is either empty or a branch that has some value of type \( A \) along with two binary trees of type \( A \) as children. Its formal definition is provided on page 3 of the appendix. For example, the definition

```haskell
Example exTree : tree nat :=
   Branch 5
      (Branch 2
         (Branch 1 Empty Empty)
         (Branch 4 Empty Empty))
      (Branch 9
         Empty
         (Branch 7 Empty Empty)).
```

represents the following binary tree:

```
      5
     /|
    2|9
   /|
  1|4|7
```

A subtree of a binary tree is either the tree itself or a subtree of its child. For example, here are all the subtrees of `exTree`:

```
Empty   1 4 7 1 4 7 1 4 7
       2 9 2 9
```

(continued…)
Give an inductive defininition of the subtree relation on binary trees, where $\textit{subtree } s \ t$ indicates that $s$ is a subtree of $t$.

Inductive $\textit{subtree} \{A : \text{Type}\} : \text{tree } A \rightarrow \text{tree } A \rightarrow \text{Prop} := $
6. (12 points) This problem asks you to translate mathematical ideas from Coq into English. In each part, your task is to find a short English summary of the meaning of a proposition defined in Coq. For example, if we gave you this definition...

\[ \text{Inductive } D : \text{nat} \to \text{nat} \to \text{Prop} := \]
\[ | D1 : \forall n, D n 0 \]
\[ | D2 : \forall n m, (D n m) \to (D n (n + m)). \]

... your summary could be “\( D m n \) holds when \( m \) divides \( n \) with no remainder.”

(a) \[ \text{Inductive } R \{X : \text{Type}\} : X \to \text{list } X \to \text{Prop} := \]
\[ | R1 : \forall x l, R x (x::l) \]
\[ | R2 : \forall x y l, (R x l) \to (R x (y::l)). \]

\( R X x 1 \) holds when:

(b) \[ \text{Inductive } R \{X : \text{Type}\} : \text{list } X \to \text{list } X \to \text{Prop} := \]
\[ | R1 : R [] [] \]
\[ | R2 : \forall x l1 l2 l3 l4, (R (l1++l2) (l3++l4)) \to (R (l1++[x]++l2) (l3++[x]++l4)). \]

\( R X 11 12 \) holds when:

(c) \[ \text{Inductive } R \{X : \text{Type}\} : \text{list } X \to \text{list } X \to \text{Prop} := \]
\[ | R1 : R [] [] \]
\[ | R2 : \forall x l1 l2 l3 l4, (R (l1++12) (l3++14)) \to (R (l1++[x]++12) (l3++[x]++14)). \]

\( R X 11 12 \) holds when:

(d) \[ \text{Definition } R (m : \text{nat}) := \]
\[ m > 1 \land (\forall n, 1 < n \to n < m \to \neg (D n m)). \]

(where \( D \) is given at the top of the page).

\( R m \) holds when:
7. **[Advanced Only]** (16 points) In two homework assignments this semester, we’ve worked with binary numerals and computations over them. On page 8 in the appendix you can find our definition of binary numerals, an \texttt{incr} operation, and a conversion function from binary to standard unary numerals.

Write a careful *informal* proof of the following theorem from *Software Foundations*:

\begin{center}
\begin{tabular}{|l|}
\hline
\textbf{Theorem:} For all binary numbers \( b \), \texttt{bin\_to\_nat} (\texttt{incr} \( b \)) is equal to \( 1 + \texttt{bin\_to\_nat} \( b \)).
\hline
\end{tabular}
\end{center}
For Reference

Numbers

Inductive nat : Type :=
    | O : nat
    | S : nat -> nat.

Fixpoint plus (n : nat) (m : nat) : nat :=
    match n with
    | O => m
    | S n' => S (plus n' m)
    end.

Notation "x + y" := (plus x y) at level 50, left associativity : nat_scope.

Fixpoint mult (n : nat) (m : nat) : nat :=
    match n with
    | O => 0
    | S n' => m + (mult n' m)
    end.

Notation "x * y" := (mult x y) at level 40, left associativity : nat_scope.

Fixpoint beq_nat (n m : nat) : bool :=
    match n, m with
    | O, O => true
    | S n', S m' => beq_nat n' m'
    | _, _ => false
    end.

Inductive le : nat -> nat -> Prop :=
    | le_n : forall n, le n n
    | le_S : forall n m, (le n m) -> (le n (S m)).

Notation "m <= n" := (le m n).

Inductive ev : nat -> Prop :=
    | ev_0 : ev 0
    | ev_SS : forall n : nat, ev n -> ev (S (S n)).

Lists

Inductive list (X:Type) : Type :=
    | nil : list X
    | cons : X -> list X -> list X.
Arguments nil {X}.
Arguments cons {X} _ _.
Notation "[ ]" := nil.
Notation "x :: l" := (cons x l) at level 60, right associativity.
Notation "[ x ; .. ; y ]" := (cons x .. (cons y nil) ..).
Fixpoint app {X : Type} (l1 l2 : list X) : (list X) :=
  match l1 with
  | [] => l2
  | h :: t => h :: (app t l2)
end.

Notation "x ++ y" := (app x y) (at level 60, right associativity).

Fixpoint In {A : Type} (x : A) (l : list A) : Prop :=
  match l with
  | [] => False
  | x' :: l' => x' = x / In x l'
end.

Regular Expressions

Inductive reg_exp (T : Type) : Type :=
  | EmptySet : reg_exp T
  | EmptyStr : reg_exp T
  | Char : T -> reg_exp T
  | App : reg_exp T -> reg_exp T -> reg_exp T
  | Union : reg_exp T -> reg_exp T -> reg_exp T
  | Star : reg_exp T -> reg_exp T.

Inductive exp_match {X: Type} : list X -> reg_exp X -> Prop :=
  | MEmpty : exp_match [] EmptyStr
  | MChar : forall x, exp_match [x] (Char x)
  | MApp : forall s1 re1 s2 re2,
    exp_match s1 re1 ->
    exp_match s2 re2 ->
    exp_match (s1 ++ s2) (App re1 re2)
  | MUnionL : forall s1 re1 re2,
    exp_match s1 re1 ->
    exp_match s1 (Union re1 re2)
  | MUnionR : forall re1 s2 re2,
    exp_match s2 re2 ->
    exp_match s2 (Union re1 re2)
  | MStar0 : forall re, exp_match [] (Star re)
  | MStarApp : forall s1 s2 re,
    exp_match s1 re ->
    exp_match s2 (Star re) ->
    exp_match (s1 ++ s2) (Star re).

Notation "s =~ re" := (exp_match s re) (at level 80).
Binary Trees

Inductive tree (A : Type) :=
  Empty : tree A
  | Branch : A → tree A → tree A → tree A.

Arguments Empty {A}.
Arguments Branch {A} _ _ _.

Binary Numbers

Inductive bin : Type :=
  | BZ : bin
  | T2 : bin → bin
  | T2P1 : bin → bin.

Fixpoint incr (m:bin) : bin :=
  match m with
  | BZ => T2P1 BZ
  | T2 m' => T2P1 m'
  | T2P1 m' => T2 (incr m')
  end.

Fixpoint bin_to_nat (m:bin) : nat :=
  match m with
  | BZ => 0
  | T2 m' => 2 * bin_to_nat m'
  | T2P1 m' => 1 + 2 * bin_to_nat m'
  end.