CSE 191: Discrete Structures
Spring 2024 Take-Home Final Exam

Matthew G. Knepley
211A Capen Hall knepley@buffalo.edu

Instructions

You have 27 hours to complete this exam from the start time, 11:00am Thursday May 9th. Thus all exams must be submitted by 2:00pm Friday May 10th. You are to turn in your exam on Autolab, exactly as you turn in homework. If you have a problem with submission, mail your Coq `.v` file directly to the instructor before the submission deadline. All exams must be submitted in Coq format.

This is an open book exam. You may use all books, notes, and other educational materials you find. You may not consult with other students. For all questions, you must submit a valid constructive proof. You may not use the following tactics:

1. `auto`,
2. `easy`,
3. `eauto`,
4. `tauto`,
5. `intuition`,
6. `lia`,
7. `nia`,
8. `now`,
9. `omega`,
10. `axiom`,
11. `Admitted`.

You are encouraged to use the `ring` tactic and the tactics suggested specifically in the problems.

Remember that if you have an algebraic identity that you want to use, for instance

$$(n+1)(n+2)+5n+1 = n^2+8n+3,$$

then you can assert that statement

```
assert (H : (n + 1) * (n + 2) + 5 * n + 1 = n * n + 8 * n + 3).
```

prove it using `ring`, and then transform your equation using

```
rewrite H.
```

Submission File

Please use this template submission file in order to get the imports and definitions right. It is also posted on the website as `FinalExamTemplate2024.v`.

```
Require Import Arith.
Lemma prob1 : forall A B C D : Prop, (A -> B) -> (A -> C) -> (B -> C -> D) -> A -> D.
Proof.
Qed.

Lemma prob2 : forall P Q : Prop, (P \/ ~ P) -> (((P -> Q) -> P) -> P).
Proof.
Qed.

Lemma prob3 : (forall P : Prop, ~ P \/ P) -> (forall P Q : Prop, ~ (P /\ Q) -> ~ P \/ ~ Q).
Proof.
Qed.

Inductive acc : nat -> Prop := acc_k : forall k, (forall y, y < k -> acc y) -> acc k.

Theorem lt_wf : forall n, acc n.
Proof.
(induction n).
 (apply acc_k).
 (intros y H).
 (inversion H).

 (induction IHn).
 clear H0.
 (apply acc_k).
 (intros y Hlt).
 (apply acc_k).
 (intros y0 Hlt0).
 (apply acc_k).
 (intros y1 Hlt1).
 (apply H).
 Require Import Arith.
 (apply le_trans with (m := y0)).
  exact Hlt1.

  (apply le_trans with (m := y)).
   (apply lt_le_weak).
   exact Hlt0.

   (apply le_S_n).
   exact Hlt.
```

```
Qed.

Definition acc_then_Px :
  forall {P : nat -> Prop} (n : nat),
    (forall x, (forall y, y < x -> P y) -> P x)
    -> acc n
    -> P n.
Proof.
  refine (fun P n f acc => acc_ind _ _ _ _).
  - intros k _ f'.
    apply f.
    exact f'.
  - exact acc.
Defined.

Definition strong_nat_ind :
  forall P : nat -> Prop,
    (forall x, (forall y, y < x -> P y) -> P x)
    -> forall x, P x.
Proof.
  refine (fun P f x => _).
  refine (acc_then_Px _ f _).
  apply lt_wf.
Defined.

Inductive div4 : nat -> Prop :=
| div4_0 : div4 0
| div4_S : forall n, rem2 n -> div4 (S (S n))
with rem2 : nat -> Prop :=
| rem2_S : forall n, div4 n -> rem2 (S (S n)).

Lemma prob4 : forall n, div4 n -> ~ rem2 n.
Proof.
Qed.

Definition relation (X Y : Type) := X -> Y -> Prop.
Definition reflexive {X: Type} (R: relation X X) := forall a : X, R a a.
Definition transitive {X: Type} (R: relation X X) := forall a b c : X,
  (R a b) -> (R b c) -> (R a c).
Definition symmetric {X: Type} (R: relation X X) := forall a b : X, (R a b) -> (R b a).
Definition antisymmetric {X: Type} (R: relation X X) := forall a b : X,
  (R a b) -> (R b a) -> a = b.
Definition equivalence {X:Type} (R: relation X X) := (reflexive R) /\ (symmetric R)
  /\ (transitive R).
Definition partial_function {X Y: Type} (R: relation X Y) := forall x : X,
```

```
  forall y1 y2 : Y, R x y1 -> R x y2 -> y1 = y2.

Lemma prob5 : ~partial_function le.
Proof.
Qed.

From Coq Require Import Bool.Bool.
From Coq Require Import Classes.RelationClasses.

Variable (choice : forall {A P} (prf : exists a : A, P a), A).
Hypothesis (choice_ok : forall {A P} (prf : exists a : A, P a), P (choice prf)).

Record Func (dom cod : Type) : Type :=
  mkFunc {
      rel : dom -> cod -> Prop
    ; total : forall x : dom, exists y : cod, rel x y
    ; functional : forall x : dom, forall y z : cod, rel x y -> rel x z -> y = z
    }.

Definition app : forall {dom cod}, Func dom cod -> dom -> cod.
Proof.
  intros dom cod f x.
  exact (choice (f.(total dom cod) x)).
Defined.

Lemma app_iff_rel : forall {dom cod} (f : Func dom cod) (x : dom) (y : cod),
    app f x = y <-> (f.(rel dom cod) x y).
Proof.
  intros dom cod f x y.
  set (s := total dom cod f x).
  unfold app.
  fold s.
  split.
  - intros app.
    rewrite <- app.
    exact (choice_ok s).
  - intros rel.
    apply (f.(functional dom cod)) with (x := x).
    + exact (choice_ok s).
    + exact rel.
Qed.

Definition by_formula : forall {dom cod : Type}, (dom -> cod) -> Func dom cod.
Proof.
  intros dom cod f.
  refine (mkFunc dom cod (fun x y => f x = y) _ _).
```

```
  - intros x.
    exists (f x).
    reflexivity.
  - intros x y z eq1 eq2.
    rewrite <- eq1.
    rewrite <- eq2.
    reflexivity.
Defined.

Lemma by_formula_ok : forall {dom cod : Type} (formula : dom -> cod) (x : dom),
    app (by_formula formula) x = formula x.
Proof.
  intros dom cod formula x.
  unfold by_formula.
  unfold app.
  simpl.
  set (ok := choice_ok
              (ex_intro (fun y : cod => formula x = y) (formula x) eq_refl)).
  symmetry.
  exact ok.
Qed.

Definition surjective {dom cod} (f : Func dom cod) := forall y : cod,
  exists x : dom, app f x = y.
Definition injective {dom cod} (f : Func dom cod) := forall x xp : dom,
  app f x = app f xp -> x = xp.

Definition f := by_formula (fun x : nat => 5 * x + 1).

Lemma prob6 : injective f /\ ~surjective f.
Proof.
Qed.

Open Scope Z_scope.
Require Import ZArith.
Require Import Znumtheory.

Lemma prob7 : forall a b c : Z, (a | c) -> (b | c) -> Zis_gcd a b 1 -> (a*b | c).
Proof.
Qed.

Definition congruence (a b c : Z) := (a | (b - c)).
Lemma prob8 : forall a : Z, equivalence (congruence a).
Proof.
Qed.
```

```
Close Scope Z_scope.
```

1 Implication

Prove that

$\forall A\, B\, C\, D \in \text{Prop}, (A \implies B) \implies (A \implies C) \implies (B \implies C \implies D) \implies A \implies D.$

for which the Coq statement is

```
Lemma prob1 : forall A B C D : Prop, (A -> B) -> (A -> C) -> (B -> C -> D) -> A -> D.
```

2 Negation and Disjunction

Prove that the Law of Excluded Middle implies the Pierce relation

$$\forall P\, Q \in \text{Prop}, \neg\,(P \vee \neg P) \implies ((P \implies Q) \implies P) \implies P,$$

following the instructions for an acceptable proof. The Coq statement is

```
Lemma prob2 : forall P Q : Prop, (P \/ ~ P) -> (((P -> Q) -> P) -> P).
```

3 Negation and Conjunction

Show that the Law of Excluded Middle implies one of DeMorgan's Laws,

$$(\forall P : \text{Prop}, \neg P \vee P) \implies (\forall PQ : \text{Prop}, \neg(P \wedge Q) \implies \neg P \vee \neg Q).$$

following the instructions for an acceptable proof. The Coq statement is

```
Lemma prob3 : (forall P : Prop, ~ P \/ P) -> (forall P Q : Prop, ~ (P /\ Q) -> ~ P \/ ~ Q)
```

4 Strong Induction

Let us divide the natural numbers into four classes: those evenly divisible by 4, those with remainder 1, remainder 2, and remainder 3. We can write an inductive type for the first and third classes as follows

```
Inductive div4 : nat -> Prop :=
| div4_0 : div4 0
| div4_S : forall n, rem2 n -> div4 (S (S n))
with rem2 : nat -> Prop :=
| rem2_S : forall n, div4 n -> rem2 (S (S n)).
```

We will prove that these two types are disjoint,

```
  Lemma prob4 : forall n, div4 n -> ~ rem2 n.
```

Strong induction is suggested for this problem, and remember that you can use the `inversion` tactic to unfold inductive definitions.

5 Relations

Prove that the relation `le` is not a partial function, where a partial function satisfies

$$\forall x : X, \forall y_1\ y_2 : Y, (R\ x\ y_1) \implies (R\ x\ y_2) \implies y_1 = y_2.$$

In Coq, you would prove

```
Lemma prob5 : ~partial_function le.
```

6 Functions

Show that the function f on the natural numbers $\mathbb{N}$ given by

$$f(n) = 5n + 2$$

in injective, but not surjective. The Coq definition of this function can be expressed as a relation,

```
Definition f := by_formula (fun x : nat => 5 * x + 1).
```

Prove that f is injective and not surjective,

```
Lemma prob6 : injective f /\ ~surjective f.
```

You will likely find it necessary to use `Nat.add_cancel_r`, `Nat.mul_cancel_l`, and `discriminate` in the first proof. You might use `discriminate` in the second step.

7 Divisibility and GCD

Prove that

$$\forall a\, b\, c \in \mathbb{Z}, (a \mid c) \implies (b \mid c) \implies 1 = \gcd(a, b) \implies (ab \mid c).$$

following the instructions for an acceptable proof. The Coq statement is

```
Lemma prob7 : forall a b c : Z, (a | c) -> (b | c) -> Zis_gcd a b 1 -> (a*b | c).
```

The `Zis_gcd_bezout` lemma can be `apply`'ed to get a `Bezout` statement, which can then be `destruct`ed, to get the Bezout relation from `Zis_gcd`.

8 Relations and Divisibility

A congruence is the statement that two integers b and c are equal modulo another integer a,

$$b = c \pmod{a}$$

which is equivalent to the statement that a divides their difference,

$$a \mid (b - c)$$

and we will encode this in Coq as

```
Definition congruence (a b c : Z) := (a | (b - c)).
```

Prove that for all integers, the integer congruence relation is an equivalence relation,

```
Lemma prob8 : forall a : Z, equivalence (congruence a).
```