1. (This is the last problem from the mid-term. As only a few got this one right, I decided to give you another chance)

(a) Suppose we decide to have letrec $x : \sigma = M$ in $N$ as a term in PCF (i.e. without desugaring it into another term that uses fix). What would be a sound typing rule for this construct?

(b) In the book, fix is defined as an operator. An alternative way is to define fix as a constructor with the syntax $(\text{fix } c : \sigma . M)$. The reduction axiom for this form of fix is:

$$\text{fix } c : \sigma . M \rightarrow M [x := \text{fix } c : \sigma . M]$$

For example, $(\text{fix } c : \text{nat} \rightarrow \text{nat}.(\lambda y : \text{nat}.y)) \rightarrow (\lambda y : \text{nat}.y)$ as $x$ is not free in $(\lambda y : \text{nat}.y)$ (i.e. the definition is not really recursive). Show how letrec $x : \sigma = M$ in $N$ can be de-sugared into a term using fix$^c$.

(c) Propose a sound typing rule for terms of the form $(\text{fix } c : \sigma . M)$.

2. Let $\Sigma = (\{\text{nat}\}, \emptyset)$ be a $\lambda^-$ signature. Let $\mathcal{A}$ be the Henkin model for $\Sigma$ where $\mathcal{A}^{\text{nat}}$ is $N$ and $\mathcal{A}^{\sigma \rightarrow \tau}$ is the uncountable set of functions from $\mathcal{A}^\sigma$ to $\mathcal{A}^\tau$, and where $\text{App } f x = f(x)$.

(a) Define $\eta = \{y \mapsto d_0\}$ where $d_0 \in \mathcal{A}^{\text{nat}}$. Show that

$$\mathcal{A}[y : \text{nat} \triangleright (\lambda x : \text{nat}.x) y] \eta = \mathcal{A}[y : \text{nat} \triangleright y : \text{nat}] \eta$$

(b) Using the equational theory for the Simply-Typed Lambda Calculus, show that the following equation is provable:

$$y : \text{nat} \triangleright (\lambda x : \text{nat}.x) y = y : \text{nat}$$

3. This problem shows that, since fields are updatable, it is generally unsound to override them with different types. Suppose $\text{PosReal} <: \text{Real}$ and let the natural logarithm function have type $\text{ln} : \text{PosReal} \rightarrow \text{Real}$. Consider the following program:

```java
class A is
    var x:Real := 0;
    method m1() is
```
subclass B of A is
  override var x:PosReal := 0;
  method m2() is
    ...
  end;
end;

procedure f() is
  var o:InstanceTypeOf(B) := new B;
  o.m1();
  o.m2();
end;

procedure g() is
  var o:InstanceTypeOf(B) := new B;
  o.m2();
  o.m1();
end;

Answer these two parts:

(a) Notice that class B overrides field x covariantly. Provide definitions for the methods m1 and m2 so that the program is typable but generates an run-time error if procedure f is executed.

(b) Same as part (a), except we assume that x has type PosReal in class A and type Real in class B, i.e. it is overriden contravariantly. Provide definitions for the methods m1 and m2 so that the program is typable but generates an run-time error if procedure g is executed.

(Hint: an example of an run-time error is the execution of ln(z) where z is a negative real, as the type of ln is PosReal→Real.)

4. Parts (a) and (b):

(a) Consider the example given on page 28 in the book (slide 10 in AC\_Chap3.pdf), and suppose we allow Vegetarian <: Person. Write a procedure f showing that, under these assumptions, it is possible for a vegetarian to eat meat (briefly explain why this happens).
(b) Suppose we have the following subtyping relations Steak $<$: Meat $<$: Food and Carrot $<$: Vegetables $<$: Food. Using the subtyping rule on arrow types, prove or disprove the following statements (for simplicity we use the initials of the types):

(a) $((F \rightarrow V) \rightarrow S) \rightarrow C <: ((F \rightarrow V) \rightarrow M) \rightarrow V$

(b) $((F \rightarrow V) \rightarrow M) \rightarrow C <: ((V \rightarrow F) \rightarrow S) \rightarrow V$

(c) $((F \rightarrow V) \rightarrow S) \rightarrow C <: ((V \rightarrow F) \rightarrow F) \rightarrow F$

(Hint: eat something before doing this exercise or you may get hungry :-).