Submission format. Put your answers into one electronic file and submit it through CourseSite. Please ensure that you file has a comment that includes your name, Lehigh network user id. Please add comments that clearly mark your solutions for each problem.

1. (5 points) For each of the types below, write down a term in System F that has that type. Also write down the typing derivation for your term.

(a) ∀X. X → X
(b) ∀X. X → X → X (write two different terms that have this type)
(c) ∀X. ∀Y. X → Y → Y
(d) ∀X. ∀Y. (X → Y) → X → Y
(e) ∀X. (X → X) → X → X (write two different terms that have this type)

2. (2 points) In class we discussed that we can use the following type to encode the type of Church Booleans in System F:

CBool = ∀X.X → X → X

Define logical or and xor in System F with respect to the above boolean type. Their types should be CBool → CBool → CBool.

3. (5 points, Appel) Perhaps you have seen the wooden dolls made in Russia that nest within each other. Create a module using a signature and structure in OCaml to model a wooden-doll factory. You will need a type doll and a type box. There will be the following operations in this module:

**baby** The client of a factory can ask for the creation of a “baby” doll, one that is solid wood and contains no other doll.

**shell** The client can return any doll d to the factory and ask for a “shell” doll, a new, hollow doll d’ that contains d.

**box** The client can pass a list of dolls back to the factory and ask for a shipping carton (box) containing them all.

**dollWeight** The weight of a baby doll is 10 grams. The weight of a shell doll containing n dolls is 10(n + 1) grams plus the weight of all the dolls contained.

**boxWeight** The weight of a box is 1000 grams plus the weight of the dolls contained. The client can ask for the weight of any box.

There are no other operations on dolls and boxes. We can define the following module type that reflects the above description of a doll factory:
module type DOLLFACTORY = sig

  type doll

  type box

  val baby : doll

  val shell : doll -> doll

  val box : doll list -> box

  val dollWeight : doll -> int

  val boxWeight : box -> int

end

• (2 points) Write a reference implementation: that is, an OCaml module that implements this module type as straightforwardly as possible. In this reference implementation, define the types doll and box as follows:

    type doll = Baby | Shell of doll

    type box = Box of doll list

• (3 points) Write a candidate implementation that implements this signature more efficiently. For each type in the signature, use as efficient a representation as possible, given the set of operations required. If you can’t think of a particularly efficient representation than at least come up with one that is significantly structurally different from your reference implementation.

Note: A baby doll weighs 10 grams. A doll shell(baby) weighs 30 grams: 20 for the shell, 10 for the baby. The doll shell(shell(baby)) weighs 60 grams: 30 for the outer shell, 20 for the inner shell, 10 for the baby.

If I create

    a = baby
    b = shell(a)
    c = baby
    d = shell(c)
    e = shell(d)

Then I can create a box with the list [b,e].

    bx = box([b;e])
You may want to go through Chapter 14 of the online book “Developing Applications with Objective Caml” before doing this problem.