Foundations of Functional Programming 2004
Paper 5 Question 10 (ACN)

I will give the function definitions needed here to show how very concise they are.

(a) Represent the number n by

```
    n f a = f(f(f...(f a))) with n applications of f.
```

Thus zero f a = a, one f a = f a.

Then amazingly easily

```
fun add n1 n2 f a = n1 f (n2 f a);
fun mult n1 n2 f a = n1 (n2 f) a;
fun ifzero n X Y = n (fn z=>Y) X;
```

(b) Turn fun f x = [[..f..]] into
val f = Y (fn f=> [[..f..]])

where fun Y f = let fun g h = f (h h) in g g end;

and Y is the standard fixed-point operator satisfying
Y f = f(Y f).

(c) I would start with

```
  fun fact n = ifzero n one (times n (fact (sub1 n)))
```

where times, ifzero are as above and sub1 is the thing I have been given. Then I just do what (b) said and get

```
fact == Y (fn fact => fn n=>
   ifzero n one (times n (fact (sub 1n))))
```

(d) The main thing here is that the definition of Y as a lambda-expression includes an application of somethihng to itself, and the ‘‘occurs test’’ in unification will reject this. So although overall Y has a valid type ’a->’a->’a in ML its definition in terms of raw lambdas will NOT typecheck.

Otherwise the arithmetic things type OK.