Functional Programming: Exercise 2

Tyng–Ruey Chuang

2007 Formosan Summer School
on Logic, Language, and Computation
July 2–13, 2007

Homework due 9:30 am, July 9, 2007.
No late homework will be accepted.

Problem 1

Complete the following definition of function `concat` so that it will return the concatenation of the lists in the input list.

```
let rec fold (base, step) list =
    match list with
          [] -> base
        | hd :: tl -> step (hd, fold (base, step) tl)

let concat ll = fold __________________________ ll

let this = concat [[1]; [2; 3]; [4; 5; 6]; [7; 8]; [9]; []]
```

When submitted to the O'Caml interpreter, you shall see

```
val fold : 'a * ('b * 'a -> 'a) -> 'b list -> 'a = <fun>
val concat : 'a list list -> 'a list = <fun>
val this : int list = [1; 2; 3; 4; 5; 6; 7; 8; 9]
```

Problem 2

Complete the following definition of function `revcat` so that it will return the reversal of the concatenation of the lists in the input list.

```
let revcat ll =
    let rec loop ll acc = ____________________
 in
    loop ll []

let that = revcat [[1]; [2; 3]; [4; 5; 6]; [7; 8]; [9]; []]
```

When submitted to the O'Caml interpreter, you shall see

```
val revcat : 'a list list -> 'a list = <fun>
val that : int list = [9; 8; 7; 6; 5; 4; 3; 2; 1]
```

Problem 3

The following definition of `nat` can be used to express all natural numbers:

```
type 'a t = Z | S of 'a
type nat = R of nat t
```

Complete the following definitions (assuming $\mathtt{u} \geq 0$):

```
let rec   fold f n = ___________________________
let rec unfold g n = ___________________________
let int2nat u = unfold ___________________________ u
let nat2int v =   fold ___________________________ v
let x = int2nat 3
let y = nat2int x
```

When submitted to the O'Caml interpreter, you shall see

```
val fold : ('a t -> 'a) -> nat -> 'a = <fun>
val unfold : ('a -> 'a t) -> 'a -> nat = <fun>
val int2nat : int -> nat = <fun>
val nat2int : nat -> int = <fun>
val x : nat = R (S (R (S (R (S (R Z))))))
val y : int = 3
```