1 - Type Inference (20 points)

1) Suppose we have the following expressions (we omit some information and we replace it with #n, where n is some positive integer).

```
a : #1 list
b : #2
let c = b :: a
let d = (1, true) :: c
```

where the last expression type checks without error.

- What is the type #1? int * bool
- What is the type #2? int * bool
- What is the type of the expression ([d])?

(int * bool) list list

2) Suppose we have the following expressions (we omit some information and we replace it with #n, where n is some positive integer):

```
a : int list
b : #1
c : #2
let (x,y) = b in (x :: a, [x + y] @ c)
```

where the last expression type checks without error.

- What is the type #1? (int * int)
- What is the type #2? int list
- What is the type of the expression ([[x]] , c :: [])?

int list list

int list list * int list list

3) Suppose x_0 could be matched as below without error

```
match x0 with bool list
| [ ]     -> [true]
| x::xs   ->  x
```

- What is the type of the variable x_0?

bool list list

- Now consider the first match for x_0, where x_0 is getting pattern matched with `[ ]`. Let us suppose that we replace `[true]` with `["true"]`. Do you think this replacement will result in some kind of error indicating that the types are no longer consistent? Answer this question either with a YES or a NO and explain why.

No, adding " will change its type from bool to string, but it will still be consistent. because X_0 will change to string list list and ["true"] will become string list.

4) Consider the following program:

```
let foo ls =
    let rec aux ls a =
    match ls with
    | [] -> a
    | hd::tl -> aux tl (hd + a)
in aux ls 0
```

- What is the type of a? Briefly explain your reasoning.

int. Because in aux ls 0.
and 0 is an int.

- What is the type signature of `foo`? Briefly explain your reasoning.

int list -> int

Because a is int, ls is int list.
Therefore, hd is int, tl is int list. so -> hd + a = int + int = int
aux will keep rec until tl = [] -> a,
so foo is int list -> int.

2 - Pattern matching (18 points)

1) Consider the following expressions

```
a: bool
b: bool
c: bool
```

true / false

TFF TFT TTT TTF
FTT FTF FFF FFT

(8) types.

Complete the following match so that every possible expression that replaces `a`, `b` and `c` in the matching statement is matched. You must ensure that you have exhausted all possible cases for `a`, `b` and `c`. Please do not use underscore in your match.

```
match (a,b,c) with
```

T, F, F →
T, F, T →
T, T, T →
T, T, F →
F, T, T →
F, T, F →
F, F, F →
F, F, T →

(4)

2) Consider the following expressions

```
a: int list * unit
b: float list
```

Complete the following match so that every possible expression that replaces a and b in the matching statement is matched, distinguishing cases for lists and tuples – no need to distinguish cases for `int` and `float`. Please do not use underscore in your match.

```
match (a,b) with
```

3) Suppose that `l` is an expression of type `(t list) list`, and consider the following pattern matching.

```
match l with
  ([]::xs) -> ...
| ((x::xs)::ys) -> ...
| [] -> ...
```

Is this match exhaustive? That is, does this match explore all the possible forms of `l`?

- ☑ Yes
- ☐ No

4) Suppose `l` is an expression of type `(int,int) list`, and consider the following pattern matching.

(int,int) list.

```
match l with
| [] -> ...
| ((x,y)::xs) -> ...
```

Is this match exhaustive? That is, does this match explore all the possible forms of `l`?

- ❑ Yes
- ☑ No

3 - Let-binding reduction (12 points)

1) Reduce the following expression to a value. Make sure that you show all the steps:

```
let x= 2 in let z= 2 + x in let w= z + z + x in w
```

2 → x; 4 → z; 4 → z; 2 → x

z = 4

x = 2 z = 4

w = 10

let z = 2+2 in let w = z+z+x in w

4

let w = 4+4+2 in w

10

2) Reduce the following expression to a value. Make sure that you show all the steps:

x = 10

```
let x= 7+3 in let (z,y)=(x + x, 3) in let
(x,u,w)=(5,y,z+z) in u + x
```

10 → x; 10 → x; y

5 → x; 3 → y; 3 → u; 5 → x

z = 20

x	u	w
5	y	z+z
	3	(x+x)+(x+x)
		(7+3) · 4 = 40

3+5 = 8

20

let (z,y) = (10+10, 3) in let

z = 20, y = 3 → (x,u,w) = (5,y,z+z) in u+x

let (x,u,w) = (5,3,40) in u+x

x = 5, u = 3, w = 40 → u+x = 3+5 = 8