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`?

`'a list`

- What is the type `#2`?

'a

- What is the type of the expression `([d])`?

(int *bool) 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 list *bool list)`

- What is the type `#2`?

`bool list`

- What is the type of the expression `([[x]] , c :: [])`?

(int list list *bool list list)

3) Suppose x_0 could be matched as below without error

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

- What is the type of the variable x_0?

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.

YES

x::xs could hold floats or even Booleans, if we change a Boolean to a string, we may get an error for the next match case. x::xs -> x or an error for a match case somewhere down the line.

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.

  a is an int because in the recursive case aux ls 0, 0 an int is placed into the parameter.

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

foo is type list because aux is called and it takes in lists in the match cases.

2 - Pattern matching (18 points)

1) Consider the following expressions

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

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
| (false,false,false)-> false
| (false,false,true)-> false
| (false,true,false)-> false
| (false,true,true)-> false
| (true,false,false)-> false
| (true,false,true)-> false
| (true,true,false)-> false
| (true,true,true)-> true
```

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
 | (a,b)->
 | (a,[])->
 | ([],b)->
 | ([],[])->
```

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`?

CHECK-> Yes

❑ No

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

```
match l with
| [] -> ...
```

```
| ((x,y)::xs) -> ...
```

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

Check-> 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
     let z=2+2 in let w=z+z+2 in w (replace x)
     let w=4+4+2 in w (replace z)
     w=10 (solve for w)
```

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

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

let (z,y) = (10+10,3) in let (x,u,w)=
(5,y,z+z) in u+x (solve and replace x)

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

u+x=3+5 (replace u and x)

u+x=8 (solve for u+x)