## **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?

#1 is of type tuple with an int and bool (int,bool)

● What is the type #2?

#2 is type int

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

d is a list of a list of type tuple of (int, bool)

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?

Tuple of ints

● What is the type #2?

Type of int list

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

**Tuple of int of int of a list**

3) Suppose x0 could be matched as below without error

**match** x0 **with**

| [ ] -> [true]

| x::xs -> x

● What is the type of the variable x0?

Xo is of type Bool list

● Now consider the first match for x0, where x0 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 because now the function will be returning a string list rather than a bool 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.

The variable a is of type a because the variable being pattern matched is ls which is an int list and I can be seen that in the in statement that when calling aux it is sent ls and 0 meaning a must be an int

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

Foo will receive an int list and return an int, since we know that ls is an int list when pattern matching and that if the list is empty the list will return a singular int a we know foos signature type goes from int list to int

## **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

(true, true, true)

|(true, true, false)

|(true, false, true)

|(true, false, false

|(false, true, true)

|(false, false, true)

|(false, true, false)

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

(( [] , () ), [])

|( [] , () ), x::x1)

|(( x::x1, () ), [])

|((x::x1, () ), y::y1)

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 it is exhaustive**

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?

**Yes it is exhaustive**

## **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

the evaluation of the expression is 10

x = 2

z = 2+x -> z = 4 or (2+2)

w = z+z+x -> 4+4+2 = 10

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

x = 7+3 🡪 10

(z,y) = (x+x, 3) 🡪 (10+10,3) or (20,3)

(x, u, w) = (5, y, z+z) 🡪 (5, 3, 40)

u + x 🡪 5+3 = 8

the expression evaluates to 8 overall