CS 320 Theory Homework #2

Due: TBD

Pattern Match

Exhaustively match all patterns of the following expressions using only the match keyword. You should match until there are only base types (int, float, bool, string). A wildcard (\_) should be used to match the right-hand side of the cons (\_::\_) pattern.

Below are three examples that attempt to illustrate what satisfies our requirements.

**WRONG EXAMPLE 1.1:**

e: (int \* bool) list

match e with

| [] -> …

| a :: \_ -> …

Wrong, because a is a tuple which is not a base type so it must be matched further.

**CORRECT EXAMPLE 1.2:**

e: (int \* bool) list

match e with

| [] -> …

| (a, b) :: \_ -> …

Correct, because the pattern (a, b) are of type int and bool respectively.

**CORRECT EXAMPLE 1.3:**

x: (int \* bool) list option

match x with

| None -> …

| Some l -> (

match l with

| []->…

| (a, b) :: \_ -> …)

Correct, because both constructors of x have been matched. In the case of the (\_::\_)pattern, the head element of type (int \* bool) has been matched against its left and right components. The components a has type int and b has type bool, which are base types.

1. Match the following expressions
   1. x: (int option \* float) option

match x with

None -> …

|

Some(None,a) -> …

|

Some(Some a,b) -> …

* 1. y: string option option list

match y with

[] -> …

|

[None] -> …

|

h::t -> …

* 1. z: int option list option

match z with

None -> …

|

Some [None] -> …

|

(Some h)::t -> …

Type Inference

1. Consider a function with polymorphic type f : ‘a -> ‘a -> ‘a.
   1. What is the type of f 0 ?

The Type of f 0 is int

* 1. What is the type of fun x -> f x x ? Briefly explain your reasoning.

Type a’ meaning whatever the type of x is it will be transitive and carry over, so is x is actually a char the entire function result type is of char

1. Consider a function with polymorphic type f : (‘a \* ‘b) -> (‘b -> ‘a) -> ‘a
   1. What is the type of f (0, true) ?

The type of f is an int

* 1. What is the type of fun x y -> f (x, y) ? Does it have the same type as f ? Briefly explain your reasoning.

No it does not the first part of the statement, fun, has two variables a’ and b’ but results in the call of function f of a tuple of a’ and b’, which we don’t know the result of

1. Consider a function with polymorphic type g : (‘a -> ‘b –> ‘b) -> ‘b -> ‘b.
   1. What is the type of g (^)?

If you are using ^ that means that g is of string type or there will be an error

* 1. What is the type of fun x -> (g g) x ? Briefly explain your reasoning.

The type of fun x would be a string assuming what we can infer from 4.1 is true that g is a string we are calling 2 g functions of g and it result in a string

1. Consider the following function.

let f (x : bool list) : int list =

match x with

| [] -> x

| hd :: tl -> 0 :: []

Is it well typed? Briefly explain your reasoning.

No it is not because the first match case is returning the given x which is bool list not an int list which is what is supposed to be returned

Higher Order Functions

In the following section, you have access to the fold\_left standard library function with the following signature.

fold\_left : (‘a -> ‘b -> ‘a) -> ‘a -> ‘b list -> ‘a

You may **not** use pattern matching or calls to other library functions. If a function has been declared in a previous problem, you may call it in future problems even if you have not worked out its exact solution. (Example: you may use 6.1 rev inside of 6.2 append, but you may not use 6.4 filter inside of 6.3 map.)

1. Implement the following standard list functions. When given the same input, they should have the same output as their standard library counterparts.
   1. rev : ‘a list -> ‘a list

let rev (a:int list): int list =

List.fold\_left(fun elem rev -> rev::elem) [] a

* 1. append : ‘a list -> ‘a list -> ‘a list

let append a b=

rev(List.fold\_left ( fun elem x ->

* 1. map : (‘a -> ‘b) -> ‘a list -> ‘b list

let map a b =

rev(List.fold\_left ( fun elem x -> (a x):: elem) [] b

* 1. filter : (‘a -> bool) -> ‘a list -> ‘a list

let aux a x b =

if (a b) then b::x else x

let filter a x =

rev(List.fold\_left (aux a ) [] x)

* 1. fold\_right : (‘a -> ‘b -> ‘b) -> ‘a list -> ‘b -> ‘b

1. Construct a function with the follow signature, using only functions declared above (6.1 - 6.5).

combinations : ‘a list -> ‘b list -> (‘a \* ‘b) list list

Such that when given two lists of lengths m and n respectively,

[ a1; a2; a3; … ; am ] : ‘a list

[ b1; b2; b3; … ; bn ] : ‘b list

It computes a nested list of all combinations of their elements pair together. They should be ordered as shown below. (Hint: In python we can do this with a nested for-loop. What corresponds to a for-loop in ocaml?)

[ [ (a1, b1); (a1, b2); (a1, b3); … ; (a1, bn) ];

[ (a2, b1); (a2, b2); (a2, b3); … ; (a2, bn) ];

[ (a3, b1); (a3, b2); (a3, b3); … ; (a3, bn) ];

…

[ (am, b1); (am, b2); (am, b3); … ; (am, bn) ] ]