COMP 302 - Assignment 2

Jeremie Poisson - McGill ID 260627104

McGill School of Computer Science – October 2015

1 Question 1

Given the following code :

```
let rec concat l = match l with
  | [] -> ""
  | x::xs ->  x ^ (concat xs)

let concat' l =
let rec conc l acc = match l with
  | [] -> acc
  | x::xs -> conc xs (acc ^ x)
in
   conc l ""
```

1.1 Proving concat and concat'

Lemma 1. *Given any given list* l,

```
    concat (l) ^ acc = conc l acc
```

Proof. Induction on l.

Base case.

`l = []`

`concat [] ^ acc`
$\Rightarrow$ `concat [] ^ acc`
$\Rightarrow$ `"" ^ acc` (*by program concat*)
$\Rightarrow$ `acc` (*by program ˆ*)
$\Leftarrow$ `conc [] acc` (*by program concat'*)

Induction step. For any given list l = x::t

Assuming the following **(induction hypothesis)**:

```
(concat t) ^ acc = conc t acc
```

we must prove

$$\texttt{(concat x::t) ^ acc} \Downarrow \texttt{conc x::t acc}$$

$\Rightarrow$ `(concat x::t) ^ acc`	
$\Rightarrow$ `x ^ (concat t) ^ acc`	(*by program concat*)
$\Rightarrow$ `(concat t) ^ (acc ^ x)`	(*by associativity of ˆ*)
$\Rightarrow$ `conc t (acc ^ x)`	(*by induction hypothesis*)
$\Leftarrow$ `conc (x::t) acc`	(*by program concat′*)

$\square$

1.2 Is my friend right ?

Indeed, my friend is right in the sense that his version of concat is tail-recursive and that mine is not. In the execution of a typical recursive function, the caller calls itself and wait until its called instance returns prior to returning its result. Therefore, calling such recursive functions creates a long execution stack where each call must wait until the last called function instance returns via the exit-point. Once the final return is reached, the stack is rolled up to the very first call to return its value. This means that the top-most function call must wait until the stack has rolled-up completely from all previous reccursive calls.

My friend's version uses an accumulator as a parameter to store the intermediate return value for each function calls. This way, function calls don't have to wait subsequent calls to return before exiting. They simply have to append the accumulator with the return value and pass on the accumulator to recursive calls. This is more efficient in a sense that function calls do not have to wait prior to execute and return its result. Moreover, there execution stack for this kind of recursion is more compact as well.