CO225 Lab 9

Ziyan Maraikar

August 29, 2014

1. Define the following following list functions using `map` and/or `fold`. Don't be concerned about your functions being "inefficient."

   (a) List length.

   (b) Replace element x with y in list l.

   Note that `map` and `fold` are in the `List` module of the standard library. So you refer to them as `List.map` and `List.fold_left`.

2. (a) Define a function `fold1` that folds a non-empty list without the need for an identity element. For example, `fold1 (+)[1; 2]` is equivalant to `1 + 2`.

   Use `fold1` to find the list maximum.

   (b) Define a function `flat_map f l` that applies the function `f` an `option list`.

   - The value form `Some 'a` elements are extracted and mapped using `f`.
   - `None` elements are simply removed from the result.

   For example, `flat_map (fun x -> x)[Some 1; None; Some 2]` will return `[1; 2]` (Note the lambda given is the identity function.)

3. The higher order function `zip f l1 l2` combines two lists elementwise using the function `f`. If the lists differ in length, it throws an exception.

```
let rec zip f l1 l2 =
  match (l1,l2) with
  | (hd1::tl1, hd2::tl2) -> (f hd1 hd2) :: zip f tl1 tl2
  | ([], []) -> []
  | (hd::tl, []) -> failwith "Lists differ in length"
  | ([], hd::tl) -> failwith "Lists differ in length"
```

   For example, `zip (+)[1; 2] [3; 4]` results in `[4; 6]`. Use `zip` to define the following functions:

   (a) A function `join` that takes two lists and returns a list of tuples. For example, `join [1;2] [3;4]` returns [(1,3); (2;4)]

   (b) A function `apply_list fl l` that takes a *list of functions* `fl` and applies them to the corresponding values in `l`. For example,

```
let sq x = x * x ;;
let fl = [sq; fun x-> sq (sq x)] ;;
apply_list fl [2; 3]
```

   should return `[4; 81]`