Assignment 2
COMP 302 Programming Languages and Paradigms

Brigitte Pientka
MCGILL UNIVERSITY: School of Computer Science

Due Date: Oct 2, 2014

Q1 [25 points] Inductive proofs (**Hand in your proofs as a pdf file q1.pdf**).

type `'a tree = Empty | Node` **of** `'a * 'a tree * 'a tree`

Q1.1 [10 points]. Consider the following program `reflect` which swaps *every* pair of children (not just the first pair).

```
let rec reflect t = match t with
| Empty -> Empty
| Node (x,left,right) -> Node (x, reflect right, reflect left)
```

Prove `reflect (reflect t)= t`.

Q1.2 [15 points]. Below we give two functions which both compute the overall size of a tree. The second function, called `size'`, computes the size of a tree tail-recursively using an accumulator.

```
let rec size m = match m with
  | Empty -> 0
  | Node(x, left, right) -> 1 + size left + size right

let rec size' m acc = match m with
  | Empty -> acc
  | Node(x, left, right) -> size' left (size' right (1 + acc))
```

Prove that for all `m:'a tree`, `size m = size' m 0`.

Q2 [60 points] Fun with trees.

Q2.1 [10 points]
Implement a function `create_tree: ('a * 'b)list -> ('a * 'b)tree` which when given a list of entries consisting of key and data creates a binary tree storing all the entries, i.e. it repeatedly inserts the entries into an empty tree.

You can rely on the function `insert e t` which we wrote in class. Recall its type: `insert: 'a * 'b -> ('a * 'b)tree -> ('a * 'b)tree`.

Use `fold_left:('a -> 'b -> 'a)-> 'a -> 'b list -> 'a` to traverse the list of entries and insert them one by one into a tree. `fold_left` is a library function and it behaves as follows:

`List.fold_left f e [b1; ...; bn]` returns `f (... (f (f e b1)b2)...)bn`.

Q2.2 [10 points]
Implement a function `tree_map: ('a -> 'b)-> 'a tree -> 'b tree` which when given a function `f` and a tree applies `f` to all the entries in the tree.

Q2.3 [5 points]
We want to delete the data from a tree which stores key - data pairs, i.e. implement a function `delete_data: ('a * 'b)tree -> 'a tree`.

Note that if the initial tree was a binary search tree the resulting tree is still a binary search tree of the same shape. Use the function function `tree_mapp` to write your function `delete_data`.

Q2.4 [15 points]
Intuitively, the `fold`-function on lists replaces every `::` by `f` and `nil` by an initial element `e` in a list. In this question, you are asked to implement an analogous function for trees.

The function `tree_fold: ('a * 'b * 'b -> 'b)-> 'b -> 'a tree -> 'b` expects a function `f:('a * 'b * 'b -> 'b)` together with an initial element `e` and a tree `t` and replaces each leaf with `e` and each node by the application of the three argument function `f`.

For example, given the following tree

```
t = Node (x0, Node (x1, Empty, Empty),
               Node (x2, Node (x3, Empty, Empty), Empty))
```

the result of `tree_fold f e t` is:

```
f(x0, f (x1, e, e), f (x2, f (x3, e, e), e))
```

Q2.5 [20 points]

The `tree_fold` function allows us to express many programs which traverse trees elegantly in one line.

(a) Re-implement the function `size : 'a tree -> int` which given a binary tree returns the number of nodes in the tree using `tree_fold`. (5 points)

(b) Implement the function `reflect : 'a tree -> 'a tree` which given a binary tree swaps the left and the right child using `tree_fold`. Note your function should swap *every* pair of children (not just the first pair). (5 points)

(c) Implement `inorder: 'a tree -> 'a list` which given a binary tree returns a list of all entries in order using `tree_fold` (10 points)

Q3 [15 points] Power generation.

Write a function `pow_series: int -> int list -> (int -> int)` which computes the power series. Given a constant c and a list of coefficients $a_0, \ldots, a_k$ return a function :

$$f(x) = a_0 + a_1 * (x - c)^1 + a_2 * (x - c)^2 + ... + a_k * (x - c)^k$$

Note that the function you return should be independently meaningful, i.e. it should contain only calls to pow, addition, subtraction or multiplication, but no other functions. Stage your function properly!