CIS 500 Software Foundations

Homework Assignment 2

More OCaml Programming

Due: Monday, September 27 by noon

The procedure for submitting your solution to this assignment is the same as for the first homework. Instructions can be found at http://www.seas.upenn.edu/~cis500/homework.html. You can find some hints about OCaml programming style at http://www.seas.upenn.edu/~cis500/resources/programming_style.html.

1 Exercise Recall the representation of binary numbers from the first assignment:

\[
\text{type bin = One | Zero | OneAnd bin | ZeroAnd bin}
\]

For example, we can represent the number 6 (written 110 in binary) as ZeroAnd (OneAnd One) (i.e., the OneAnd and ZeroAnd constructors represent least-significant bits).

- Write a function \text{int_of_bin: bin \rightarrow int} that returns the natural number that one of these terms represent.
- Write a function \text{bin_of_int: int \rightarrow bin} that returns the bin representing the integer argument.
- Write the function \text{length : bin \rightarrow int} that returns the length of a given term when viewed as a bitstring. For example, it must be that \text{length(Zero)} = 1. You may count any leading zeros.
- Define the function \text{zeros : bin \rightarrow int} that returns the number of zeros that a given term contains when viewed as a bitstring.

2 Exercise Implement functional queues in OCaml. Your implementation must define a type for queues, an exception (called Empty) to use when removing elements from an empty queue, and the following operations. Queues should be polymorphic over the type of elements that they contain.

\[
\text{type 'a queue}
\]

\[
\text{exception Empty}
\]

(* An empty queue. *)

\[
\text{empty : 'a queue}
\]

(* Return a queue that adds the element to the back of the given queue. *)

\[
\text{add : 'a \rightarrow 'a queue \rightarrow 'a queue}
\]

(* Remove the element from the front and return it paired with the new queue. If the queue is empty, raise the exception "Empty". *)

\[
\text{take : 'a queue \rightarrow 'a * 'a queue}
\]

(* Calculate the number of elements in the queue. *)

\[
\text{length : 'a queue \rightarrow int}
\]

Note: Your code must be purely functional, written using the constructs that we have discussed in class and in recitation. You may not use library routines or assignment to implement this data structure.

3 Exercise The following data structure can be used to represent binary trees:
type 'a tree = Nil | Node of 'a * 'a tree * 'a tree

- Write a function height: 'a tree -> int that computes the height of the tree. Your function should behave like this:
  
  # height Nil;
  - int = 0
  # height (Node("root", Nil, Nil));
  - int = 1

- Write a function isPerfect: 'a tree -> bool that returns true if and only if the tree is a “perfect tree”. A perfect tree of height n contains 2^n – 1 values (i.e., Nodes). That is, every level is completely filled.

- Write a function isFull: 'a tree -> bool that returns true if and only if the tree is a “full tree”. A full tree is one in which every Node has either 0 or 2 Nil children.

- Write a function isComplete: 'a tree -> bool that returns true if and only if the tree is a “complete” tree. A complete tree is a tree in which every level, except possibly the deepest, is completely filled and at the deepest level of the tree, all nodes must be as far to the left as possible.

- In this question you will have to implement functions that traverse the tree, read-off the values stored in the nodes of the tree and output them in a list.

  Recall that in inorder traversal, the left subtree is visited first, then the current node, and then the right subtree. In postorder traversal, the current node is visited after its subtrees are visited. In preorder traversal the current node is visited before its subtrees. Finally in breadth-first search we visit all the nodes of a given depth; then proceed with all the nodes of one level deeper and so on.

  You are asked to implement all those traversal functions. Each function should return a list of the values stored in the nodes of the tree. You might find the functional queue data structure of previous exercise useful.

4 Exercise

- The forall function takes a predicate p (a one-argument function returning a boolean) and a list l and checks whether p returns true when applied to every element of l.

  # forall (fun x -> x >= 3) [10;11;55];
  - bool = true

  # forall (fun x -> x >= 3) [5;1;7;9];
  - bool = false

  Write forall as a recursive function.

- Rewrite forall as compactly as possible (e.g., using fold).

- Can the hd function be implemented in terms of map, fold, etc.?

- [Optional and challenging] How about tl?

5 Debriefing

1. How many hours did you spend on this assignment?

2. Would you rate it as easy, moderate, or difficult?

3. Did you work on it mostly alone, or mostly with your study group?

4. How deeply do you feel you understand the material it covers (0%–100%)?

5. Any other comments?