Programming Assignment 5

Stacks and the Towers of Hanoi

Submission Deadline: 11:55PM, 29th October 2014

Submissions only via Moodle

A *stack* is a commonly used "abstract data structure" for storing elements in a *Last-In First-Out* (LIFO) discipline. In well-typed languages, stacks can contain only one kind (type) of data elements. However, the stack construction (like sets and lists) is generic in what type of element it may hold.

Abstractly, a stack is created and manipulated by the following operations. Operations:

- `create: unit -> 'a stack`
- `push: 'a -> 'a stack -> 'a stack`
- `isEmpty: 'a stack -> bool`
- `size: 'a stack -> int`
- `top: 'a stack -> 'a`
- `pop: 'a stack -> 'a stack`

The first two operations are abstract "constructions" of a stack (they build up a stack). The next two are "(pure) observer functions", since they can be used to observe some property of the stack (e.g., the number of elements in the stack). The last two functions are called *deconstructors*, since they allow us to take a stack apart (e.g., by seeing its topmost element).

Let a be any element of type `'a`, and S be any stack of type `'a stack`. Stacks satisfy the following axioms:

1. `isEmpty(create()) = true`
2. $\forall a, S$: `isEmpty(push(`a`,` S`) = false`
3. `size(create()) = 0`
4. $\forall a, S$: `size(push(`a`,` S`) = 1+size(`S`)`
5. $\forall a, S$: `top(push(`a`,` S`) =` a
6. $\forall a, S$: `pop(push(`a`,` S`) =` S

You need to cater to possible exceptional situations.

This assignment is to make you read the OCaml tutorials and manuals and learn more about OCaml's module facilities, namely,

- *Signatures*, which specify the components of a module but do not give details of how they are implemented. Signatures are to modules what types are to values and functions.

- Modules: a collection of related type(s), function and value definitions
- Functors: Parameterized modules, which are dependent on other modules (or types or values). The relationship of functors to modules is somewhat similar to what functions are to values.
- Modules respecting signatures (similar to values being of a particular type).

You need to

1. Write the signature STACK of a stack package **(To be checked at demo time).**
2. Represent stacks using lists and document the representational invariant.
3. Implement the stack operations as a module.
4. Check that your stack module respects the STACK signature. **(To be checked at demo time)**
5. Give appropriate test data to demonstrate that all operations satisfy the axioms listed **(To be checked at demo time)**
6. Make the stack module parametric in the type of elements (write it as an OCaml functor which gives a different type of stack module depending on the type of elements) **(To be checked at demo time)**

In the second part, you will use your stack implementation to program the Towers of Hanoi puzzle. This is supposed to have been a game devised by monks in a Vietnamese temple.

- There are 3 sticks with n rings on them.
- Each ring has a different diameter, say $1 \ldots n$.
- Initially all rings are in sorted order on Stick 1, the smallest (Ring 1) on top and largest (Ring n) at the bottom. Sticks 2, 3 are empty.
- Task: move all the rings from Stick 1 to Stick 2 (using Stick 3 as a temporary)
- Constraint 1: *Only the topmost ring can be moved at a time from any stick, and placed on any other stick*
- Constraint 2: *You are never allowed on any stick to have a larger ring anywhere above a smaller ring*, e.g., Ring 4 cannot be above Ring 2. However Ring 2 can be directly above Ring 4.

The state of the towers is given as three stacks, representing the rings on each of the sticks. A single `move` is specified by mentioning the number of the stick from which the topmost ring is moved and the number of the destination stick.

```
type move = FromTo of int * int;;
```

The puzzle is solved recursively, by moving $m \leq n$ disks from one stick to another, moving the $(m+1)^{th}$ disk and then moving the m disks to the desired stick.
Write a recursive program `hanoi: (int * int * int * int * int stack* int stack * int stack) -> (move * int stack* int stack * int stack) list`. `hanoi`$(m, i, j, k, s_1, s_2, s_3)$ lists the moves and the resulting stick configurations involved when moving m disks from Stick i to Stick j using Stick k as a helper stick. A function call `hanoi(2, 1, 3, 2, [1;2;3;4]; [5]; [])` (move 2 rings from Stick 1 to Stick 3 via Stick 2) yields `[ ( Fromto(1, 2), [2;3;4], [1;5], [ ]); ( Fromto(1, 3), [3;4], [1;5], [2]); ( Fromto(2, 3), [3;4], [5], [1;2]) ]`
You must use your Stack package. You must document your programs adequately, and test your program. You should also analyse the time and space required for solving the puzzle.
Demo instruction: You will be given test inputs and your function should solve them correctly.