Functional Programming: Exercise 3

Tyng–Ruey Chuang Max Schäfer

2007 Formosan Summer School
on Logic, Language, and Computation
July 2–13, 2007

Homework due 9:30 am, July 13, 2007.
No late homework will be accepted.
You must turn in complete listing of your code. No exception.

Note: You need not feel compelled to complete all the problems. Do as many as you can.

Problem 1

Complete the following code about the stack and queue data types. Note that you may want to first define a fold function for stacks so that it can be used to define the pop function for queues. Try a few examples using the modules you have just defined.

```
module type STACK =
sig
  type 'a t
  val empty: 'a t
  val push: 'a -> 'a t -> 'a t
  val pop: 'a t -> ('a * 'a t) option
end

module type QUEUE = STACK
module type S2Q = functor (S: STACK) -> QUEUE

module MakeQueue: S2Q = functor (S: STACK) ->
struct
  type 'a t = 'a S.t * 'a S.t
  let empty = (S.empty, S.empty)
  let push elm (front, rear) = (front, S.push elm rear)
  let pop (front, rear) = _________________________________
end
```

```
module S =
struct
  ----------------------
end

module Q = MakeQueue(S)

let q = Q.push 2 (Q.push 1 Q.empty)
```

Problem 2

Given the following data type definition `'v formula` for formulas of propositional logic (abstracted over a type variable `'v` of propositional letters), define a function `pl` to gather all unique propositional letters in a formula to a list (whose type is `'v list`).

```
type 'v formula =
    | PL of 'v
    | Bot
    | Con of 'v formula * 'v formula
    | Dis of 'v formula * 'v formula
    | Imp of 'v formula * 'v formula
```

You may want to first define a function `merge` that takes two lists and return as a result the list of all unique elements appearing in the two input lists. For your reference, the types of `pl` and `merge` are the following:

```
val merge : 'a list -> 'a list -> 'a list = <fun>
val pl : 'a formula -> 'a list = <fun>
```

Hint: You can use functions `fold_left` and `mem` in O'Caml's `List` module.

Problem 3

Given the following module type definition `BOOLEAN_ALGEBRA`:

```
module type BOOLEAN_ALGEBRA =
sig
  type b
  val cup : b * b -> b
  val cap : b * b -> b
  val inv : b -> b
  val zero : b
  val one : b
end
```

Complete the following parametric module `Evaluator`:

```
module Evaluator (BB : BOOLEAN_ALGEBRA) =
struct
  let rec eval phi v = ____________________________
  let satisfies phi v = (eval phi v) = BB.one
end
```

Note that function `eval` takes as input a formula `phi` and an environment `v` (which maps propositional letters to values of type BB.b) and returns as the result the value of `phi` under environment `v`. You can assume that environment `v` is always well-defined.

For your reference, O'Caml shall infer module `Evaluator` to have the following module type:

```
module Evaluator :
  functor (BB : BOOLEAN_ALGEBRA) ->
    sig
      val eval : 'a formula -> ('a -> BB.b) -> BB.b
      val satisfies : 'a formula -> ('a -> BB.b) -> bool
    end
```

Hint: You may want to try out your definition of `Evaluator` on following module `TruthValues`:

```
module TruthValues : (BOOLEAN_ALGEBRA with type b = bool) =
struct
  type b = bool
  let cup (x, y) = x || y
  let cap (x, y) = x && y
  let inv x = not x
  let zero = false
  let one = true
end

module TruthValueEvaluator = Evaluator(TruthValues)
```