CS 784 Programming Languages
Fall 2011
Home Work 2

---

1. (40+20 points) The ADT known as Map (Cf. java.util.Map) has the following operations described only informally here:
   i. map: Yields the empty map.
   ii. clear: Takes a map m and yields the empty map.
   iii. containsKey: Takes a map m and a key k as input, and checks if m has k.
   iv. containsValue: Takes a map m and a value v as input, and checks if m has v bound to some key.
   v. equals: Takes two maps m and n as input, and checks if m and n are equivalent, that is, contain similar bindings.
   vi. get: Takes a map m and a key k as input, and yields the value bound to k in m.
   vii. isEmpty: Takes a map m as input, and checks if m is empty.
   viii. put: Takes a map m, a key k, and a value v as input, and yields a new map that behaves like m except that k is bound to v.
   ix. remove: Takes a map m and a key k as input, and yields a new map which has k deleted from m (if it were present).
   x. size: Takes a map m as input, and yields the number of key-value pairs in m.

   1. (40 points) Specify the syntax and semantics of the ADT Map using algebraic approach. Include a proof-like argument supporting the correctness of your specs.
   2. (20 points) Give an implementation of the ADT Map in Scheme. Include at least 5 thorough test cases.

   ADT specs are written in functional style and as such do not support assignments. For example, put(m,k,v) does not change the state of the existing map m. Instead, conceptually, it creates a new map by "cloning" m and then modifying the copy by inserting the new k-v binding. Note further that: (a) The informal spec does not rule out (k, v1) and (k, v2) to be present in the map, unless we take the "the" in get-definition seriously, and we do. (b) Is get(m, k) always defined? (c) Some times put(m, k, v) = m, and hence size(put(m,k,v)) = 1 + size(m) is not always true.

2. (40 points) This problem is essentially EOPL3 Exercise 3.26 (EOPL2 Exercise 3.27). In our data-structure representation of procedures, we have kept the entire environment in the closure. But of course all we need are the bindings for the free variables. Modify the representation of procedures to retain only the free variables.

   Start with the interpreter given in the source code files of EOPL3 chapter3/proc-lang/. Consider the representation for closure described here.

```
(define closure
  (lambda (ids body env)
    (let ((freevars (set-diff (free-vars body) ids)))
      (let ((saved-env
              (extend-env
                freevars
                (map
                  (lambda (v)
                    (apply-env env v))
```

```
                freevars)
              (empty-env))))
      (lambda (args)
        (eval-expression body
          (extend-env ids args saved-env)))))))
```

where `set-diff` takes the difference of two sets. This is called the *flat closure* representation. The environment of such a closure consists of exactly one rib comprising its free variables and their values.

In order for me to better grasp the changes you made, comment out the code you are modifying, rather than deleting it. Include test calls and outcomes in your solution exercising all the constructs. Include at least 10 new tests in additions to those already in the given source code of the book.

Recall that semicolon begins a comment that goes to the end of line. Multi-line nested comments are done using #| and |#.

What to hand-in?

Submit on `gandalf.cs.wright.edu` as follows:

```
% ~pmateti/CS784/turnin HW2 ReadMe.txt problem1-1.txt problem1-2.scm problem2.scm
```
